Oracle® Fusion Middleware

Developing Applications with Oracle JDeveloper

14*c* (14.1.2.0.0)
F39367-01
December 2024

Oracle Fusion Middleware Developing Applications with Oracle JDeveloper, 14*c* (14.1.2.0.0)

F39367-01

Copyright © 2024, Oracle and/or its affiliates.

Primary Author: Oracle Corporation

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software, software documentation, data (as defined in the Federal Acquisition Regulation), or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, then the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs (including any operating system, integrated software, any programs embedded, installed, or activated on delivered hardware, and modifications of such programs) and Oracle computer documentation or other Oracle data delivered to or accessed by U.S. Government end users are "commercial computer software," "commercial computer software documentation," or "limited rights data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, reproduction, duplication, release, display, disclosure, modification, preparation of derivative works, and/or adaptation of i) Oracle programs (including any operating system, integrated software, any programs embedded, installed, or activated on delivered hardware, and modifications of such programs), ii) Oracle computer documentation and/or iii) other Oracle data, is subject to the rights and limitations specified in the license contained in the applicable contract. The terms governing the U.S. Government's use of Oracle cloud services are defined by the applicable contract for such services. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle®, Java, MySQL, and NetSuite are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Inside are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Epyc, and the AMD logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information about content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services unless otherwise set forth in an applicable agreement between you and Oracle. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services, except as set forth in an applicable agreement between you and Oracle.

Contents

Preface

Audience li
Documentation Accessibility li
Diversity and Inclusion li
Related Documents li
Conventions liii

What's New in This Guide

New and Changed Features for 14c (14.1.2.0.0) liiii

1 Introduction to Oracle JDeveloper

About Oracle JDeveloper 1-1
Oracle JDeveloper Information Resources 1-1
Configuring Proxy Settings 1-2
- Using an Automatic Configuration Script for Proxy Settings 1-3

Migrating to Oracle JDeveloper 14c 1-3

2 Oracle JDeveloper Accessibility Information

Using a Screen Reader and Java Access Bridge with Oracle JDeveloper 2-1
Oracle JDeveloper Features that Support Accessibility 2-1
- Keyboard Access 2-1
- Screen Reader Readability 2-3
- Flexibility in Font and Color Choices 2-3
- No Audio-only Feedback 2-3
- No Dependency on Blinking Cursor and Animation 2-4
- Screen Magnifier Usability 2-4
- How to Change the Editor or Tabbed View of a File 2-4
- How to Read Text in a Multi-line Edit Field 2-4
- How to Read the Line Number in the Source Editor 2-4
- How to Access Exception Stack HTML Links and Generated Javadoc Links in the Log Window 2-4

Recommendations for Customizing Oracle JDeveloper 2-5
How to Customize the Accelerators Keys 2-5
How to Pass a Conflicting Accelerator Key to Oracle JDeveloper 2-5
How to Change the Look and Feel of the IDE 2-5
How to Customize the Fonts in Editors 2-5
How to Customize Syntax Highlighting 2-5
How to Display Line Numbers in Editors 2-5
How to Change the Timing for Code Insight 2-5
How to Specify the Columns in the Debugger 2-6
Highly Visual Features of Oracle JDeveloper 2-6

3 Working with Oracle JDeveloper

About Working with Oracle JDeveloper 3-1
Working with JDeveloper Roles 3-1
How to Change the JDeveloper Role 3-2
How to Manage JDeveloper Features and Installed Updates 3-2
Working with Windows in the IDE 3-3
How to Maximize Windows 3-3
How to Minimize and Restore Dockable Windows in the IDE 3-4
How to Dock Windows in the IDE 3-4
How to Close and Reopen Dockable Windows in the IDE 3-5
Opening a Closed Window 3-5
Closing an Open Window 3-5
How to Restore Window Layout to Factory Settings 3-6
Keyboard Navigation in JDeveloper 3-6
How to Work with Shortcut Keys in the IDE 3-6
Loading Preset Keyboard Schemas 3-6
Viewing JDeveloper Commands and Associated Keyboard Shortcuts 3-7
Redefining a Keyboard Shortcut for a Command 3-7
Importing and Exporting Keyboard Schemas 3-8
Common Navigation Keys 3-8
Navigation in Standard Components 3-9
Navigating Complex Controls 3-14
Navigation in Specific Components 3-19
Customizing the IDE 3-24
How to Change the Look and Feel of the IDE 3-24
How to Customize the General Environment for the IDE 3-25
How to Customize the Compare Window in the IDE 3-25
How to Customize the Components window 3-25
How to Add a Page to the Components Window 3-25
How to Add a JavaBeans Component to the Components Window 3-26

How to Remove a Page from the Components Window 3-27
How to Remove a Component from the Components Window 3-27
How to Change Roles in JDeveloper 3-28
How to Associate File Types with JDeveloper 3-28
Working with the Resources Window 3-28
Using the Resources Window 3-29
How to Open the Resources Window 3-29
How to Refresh the Resources Window 3-29
Working with IDE Connections 3-29
Searching the Resources Window 3-31
How to Perform a Simple Search 3-31
How to Perform an Advanced Search 3-31
How to Stop and Save a Search 3-32
Filtering Resources Window Contents 3-32
Importing and Exporting Catalogs and Connections 3-32
Working with Resources Window Catalogs 3-33
Creating Catalogs 3-33
Renaming Catalogs 3-34
Working with Catalog Folders 3-34
How to Create Folders 3-34
How to Create Dynamic Folders 3-34
How to Add Resources to a Catalog 3-34
Working with Source Files 3-35
Working with Index Data 3-35
Using the Source Editor 3-35
Features Available From the Context Menu 3-38
Using Mini-Maps 3-40
Using Stepping Margin 3-40
Using Multi Cursor 3-41
Using Hyperlinking with Javadoc Comments 3-41
Using the Find and Replace Toolbar 3-42
How to Set Preferences for the Source Editor 3-42
How to Set Indentation Size for the Source Editor 3-43
How to Set Fonts for the Source Editor 3-43
How to Set Caret Behavior for the Source Editor 3-44
How to Set Display Options for the Source Editor 3-44
How to Set Line Gutter Behavior for the Source Editor 3-44
How to use the Save as HTML parameter in the Source Editor 3-45
How to Set Options for Syntax Highlighting in the Source Editor 3-45
How to Set Bookmark Options for the Source Editor 3-45
How to Customize Code Templates for the Source Editor 3-46
How to Manage Source Files in the Editor Window 3-47

How to Change the View of a File 3-47
How to Navigate Between Open Files in the Editor Window 3-47
How to Display the List of All Currently Open Files 3-48
How to Access a Recently Opened File 3-48
How to Manage Multiple Editors for a File 3-49
How to Work With Multiple Files 3-49
How to Quickly Close Files in the Editor Window 3-50
Working with Mouseover Popups 3-51
How to Locate a Source Node in a Window such as the Applications Window, Databases Window, Applications Server Window 3-52
How to Set Bookmarks in Source Files 3-52
How to Edit Source Files 3-52
How to Open Source Files in the Source Editor 3-52
How to Edit Source Code with an External Editor 3-53
How to Insert a Code Snippet from the Components Window into Source Files 3-53
How to Record and Play Back Macros in Source Files 3-54
How to Create Tasks 3-55
How to Compare Source Files 3-55
How to Revert to the Last Saved Version of a File 3-56
How to Search Source Files 3-56
How to Search Text in an Open Source File 3-56
How to Search for a Single Text String 3-57
How to Search All Files in a Project or Application 3-57
How to Print Source Files 3-57
Reference: Regular Search Expressions 3-58
Working with Extensions 3-58
How to Install Extensions with Check for Updates 3-58
How to Install Extensions from the Provider's Web Site 3-59
How to Install Extensions Directly from OTN 3-59
How to Install Extensions Using the JDeveloper dropins Directory 3-59
Using the Online Help 3-60
Using the Help Center 3-60
How to Open the Online Help 3-61
How to Search the Documentation 3-61
How to Add Bookmarks to the Favorites Page 3-63
How to Customize the Online Help Display 3-63
How to Open and Close Multiple Help Topics 3-64
How to Print Help Topics 3-64
Common Development Tools 3-65
Application Overview 3-65
Checklist 3-65
File Summary Pages 3-66

File List 3-67
File List Tab Header 3-67
Search Criteria Area 3-67
Search Results Table 3-68
Compare Window 3-69
Toolbar 3-69
Source and Target Areas 3-69
Applications Window 3-69
Applications Window Toolbar 3-69
Application Operations 3-71
Projects Panel Operations 3-71
Application Resources Panel Operations 3-72
Data Controls Panel Operations 3-72
Recent Files Panel Operations 3-72
Application Servers Window 3-73
Structure Window 3-74
Structure Window Toolbar 3-74
Structure Window Views 3-75
Applications Window - Data Controls Panel 3-76
Log Window 3-76
Issues Window 3-77
Documents Dialog 3-78
Dependency Explorer 3-79
Adding External Tools to JDeveloper 3-79
How to Find All External Programs Supported by JDeveloper 3-79
How to Add Access to an External Program from JDeveloper 3-79
How to Change the Appearance of an External Program 3-80
Working with Tasks 3-80
About Task Repositories 3-80
Working with Tasks 3-81
Finding and Opening Tasks 3-81
Creating and Saving Task Queries 3-81
Reporting New Tasks 3-82
How to Add a Task Repository 3-82
Working with the Tasks Window 3-83
How to View Tasks 3-83
How to Organize Tasks 3-84

4 Getting Started with Developing Applications with Oracle JDeveloper

About Developing Applications with Oracle JDeveloper 4-1
Creating Applications and Projects 4-1

How to Create an Application 4-2
How to Create a Custom Application 4-2
How to Create a Project 4-3
Creating a New Custom Project 4-3
Managing Applications and Projects 4-3
How to Open an Existing Application 4-4
How to Open an Existing Project 4-4
How to Quickly Add Items to a Project Using the New Menu 4-4
How to Import Existing Source Files into JDeveloper 4-5
How to Import Existing Files into a New JDeveloper Project 4-5
How to Import a WAR File into a New JDeveloper Project 4-6
How to Import an EAR File into a New JDeveloper Application 4-6
How to Import Files into a Project 4-7
Managing Folders and Java Packages in a Project 4-8
How to Create a Folder or Java Package 4-8
How to Delete a Folder or Java Package 4-8
How to Manage Working Sets 4-8
How to Group Objects into a Working Set 4-9
How to Create a Working Set by Defining File and Directory Filters 4-9
How to Create a Working Set From Search Results in the Log Window 4-9
How to Identify the Current Working Set 4-10
How to Change the Active Working Set 4-10
How to Edit Files and Projects in a Working Set 4-10
How to Show All Files in the Applications window 4-10
How to Run and Debug a Working Set 4-10
How to Browse Files in JDeveloper Without Adding Them to a Project 4-11
How to View an Archive 4-11
How to View an Image File in JDeveloper 4-12
Setting Default Project Properties 4-12
How to Set Default Project Properties 4-12
How to Set Properties for Individual Projects 4-13
How to View or Change the Current Output Path for an Individual Project 4-13
How to Set the Target Java SE for a Project 4-13
How to Manage Project Dependencies 4-14
How to Associate Features with a Project Via a Template 4-14
How to Associate Features with an Individual Project 4-15
How to Set Javadoc Properties for a Project 4-15
How to Manage Libraries 4-15
How to Add Application-level Libraries and Classpaths 4-16
How to View the Current Libraries in a Project 4-16
How to Add an Existing Library to a Project 4-17
How to Create a New Library and Add it to a New Project 4-17

How to Edit an Existing Library in a Project 4-17
How to Remove Libraries from a Project 4-18
How to Import Libraries or Java SEs Outside the Project Scope 4-18
How to Create Libraries or Java SEs Outside the Project Scope 4-19
How to Edit Libraries or Java SEs Outside the Project Scope 4-19
How to Delete Libraries or Java SEs Outside the Project Scope 4-19
How to Manage Application and Project Templates 4-20
How to Define a New Application Template 4-20
How to Define a New Project Template 4-21
How to Share Application and Project Templates 4-21
How to Edit an Existing Application or Project Template 4-21
How to Delete an Existing Application or Project Template 4-22
How to Manage File Templates 4-22
Available File Template Variables 4-26
How to Save an Application or Project 4-26
How to Save an Individual Component or File 4-27
How to Rename an Application, Project, or Individual Component 4-27
How to Relocate an Application, Project, or Project Contents 4-28
How to Close an Application, Project, or Other File 4-29
How to Remove a File from a Project 4-29
How to Remove a Project from an Application 4-30
How to Remove an Application 4-30

5 Developing Applications Using Modeling

About Modeling with Diagrams 5-1
UML Diagrams 5-1
Business Services Diagrams 5-2
Transformations 5-2
Creating, Using, and Managing Diagrams 5-2
Creating a New Diagram 5-4
Working with Diagram Elements 5-4
How to Locate an Element on a Diagram 5-5
How to Select Specific Elements on a Diagram 5-5
How to Select All Elements on a Diagram 5-5
How to Select All Elements of the Same Type 5-5
How to Deselect an Element in a Group of Selected Elements 5-5
How to Group Elements on a Diagram 5-6
How to Manage Grouped Elements 5-6
How to Change Semantic Properties 5-6
How to Change Element Properties Using the Properties window 5-6
How to Change the Element Color or Font 5-7

How to Change the Visual Properties of New Diagram Elements 5-7
How to Copy Visual Properties 5-8
How to Resize Elements on a Diagram 5-8
How to Display Related Classes on a Diagram 5-8
How to Move Diagram Elements 5-9
How to Undo the Last Action on a Diagram 5-9
How to Copy Elements to Another Diagram 5-9
How to Rename a Diagram 5-10
How to Publish a Diagram as an Image 5-10
How to Setup a Page for Printing 5-10
How to Set the Area of a Diagram to Print 5-10
How to See a Preview of Your Page Before Printing 5-11
How to Clear a Diagram Print Area 5-11
How to Zoom in and Out of a Diagram 5-11
How to Display an Entire Diagram 5-11
How to Display the Selected Elements at the Maximum Size 5-11
How to Display a Diagram in its Original Size 5-11
How to Delete a Diagram 5-11
Working with Diagram Layout 5-12
How to Use a Hierarchical Diagram Layout 5-12
How to Use Layout Edges on a Diagram 5-13
How to Use a Symmetrical Diagram Layout 5-14
How to Use an Orthogonal Diagram Layout 5-14
How to Use a Grid Diagram Layout 5-15
How to Lay Out Diagram Elements 5-15
How to Lay Out Diagrams Using the Grid 5-16
How to Distribute Diagram Elements 5-16
How to Align Diagram Elements 5-16
Working with Diagram Nodes 5-17
How to Create a Node on a Diagram 5-17
How to Create Internal Nodes on a Diagram Element 5-17
Working with Diagram Edges 5-18
How to Hide Edges on a Diagram 5-18
How to Show Hidden Edges on a Diagram 5-19
How to Show all Hidden Edges on a Diagram 5-19
How To List All Hidden Edges Together in the Structure Window 5-19
How to Change Crossing Styles on a Diagram 5-19
Annotating Your Diagrams 5-19
How to Add a Note to a Diagram 5-19
How to Attach a Note to an Element on a Diagram 5-20
How to Change the Font on a Note 5-20
Storing Diagrams 5-20

Using UML 5-20
Creating UML Elements Off a Diagram 5-21
Storing UML Elements Locally 5-21
Using UML Profiles 5-21
How to Create a Profile 5-22
How to Export a Profile 5-22
How to Add a Profile 5-22
How to Apply a Profile to a UML Package 5-23
Importing and Exporting UML 5-24
How to Import and Export UML Models Using XMI 5-24
How to Export UML Models as XMI 5-25
Typical Error Messages When Importing 5-26
Using MOF Model Libraries 5-27
How to Create an XMI Catalog File 5-28
Using Transformations 5-29
Transformation Types 5-29
UML-Java Transformation 5-30
Transform UML to Java 5-30
Transform Java to UML 5-30
UML-Offline Database Transformation 5-31
Transform an Offline Database Diagram to UML 5-31
Transform UML to Offline Database Objects 5-32
Transform UML Classes on a Diagram to an Offline Database 5-33
Set Hierarchy Options 5-33
Transformation Settings 5-36
Reuse Transform Settings 5-36
Using DatabaseProfile for UML Transformations 5-37
Use DatabaseProfile to Transform a Class Model 5-38
Logging Options 5-39
UML-ADF Business Components Transformation 5-39
Modeling with UML Class Diagrams 5-39
Creating a UML Class Diagram 5-41
Working with the Class Diagram Features 5-41
How to Create Classifiers, Constraints, and Packages 5-42
How to Create Attributes 5-42
How to Add Nested Classes and Nested Interfaces 5-43
How to Add Attributes and Operations 5-43
How to Hide Attributes and Operations 5-43
How to Add Generalizations, Realizations, and Associations 5-44
Modeling with Activity Diagrams 5-44
Working with the Activity Diagram Features 5-45
How to Create an Activity Diagram 5-46

How to Create Initial and Final Nodes 5-46
How to Show Partitions 5-46
Modeling with Sequence Diagrams 5-47
Working with the Sequence Diagram Features 5-47
How to Add and Create a Sequence Diagram 5-48
How to Start a Sequence Tracer 5-49
How to Automatically Layout Elements in an Interaction 5-49
How to Add Lifelines and Classifiers 5-49
How to Create a Synch Call 5-49
How to Work With Execution Specifications 5-50
How to Add a Create Message 5-50
How to Create a Delete Message 5-50
How to Create a Reply Message 5-50
How to Create an Async Call 5-51
Using Combined Fragments 5-51
Modeling with Use Case Diagrams 5-53
Working with the Use Case Diagram Features 5-54
Getting A Closer Look at the Use Case Diagram Elements 5-55
How to Add a Subject to a Use Case Diagram 5-56
How to Create Actors and Use Cases 5-56
How to Represent Interactions Between Actors and Use Cases 5-57
How to Represent Relationships Between Use Cases and Subjects 5-57
Exporting a Use Case Model for the First Time 5-57
Exporting a Changed Use Case Model 5-59
Importing a Use Case Model from a Set of HTML Files 5-60
Editing the HTML Files 5-60
Importing from HTML files 5-63
Modeling with Profile Diagrams 5-65
Modeling with Java Class Diagrams 5-66
How to Create Java Classes, Interfaces and Enums 5-66
How to Model Inner Java Classes and Interfaces 5-67
Modeling Composition in a Java Class Diagram 5-67
Modeling Inheritance on a Java Class Diagram 5-68
Extending Modeled Java Classes 5-69
Implementing Modeled Java Interfaces 5-69
Modeling Java Fields and Methods 5-69
Refactoring Class Diagrams 5-70
Modeling with EJB Diagrams 5-71
Working with EJB/JPA Modeling Features 5-71
Create a Diagram of EJB/JPA Classes 5-72
How to Model EJB/JPA Relationships 5-74
Reference Between Beans 5-74

Properties on Modeled Beans 5-74
Methods on Modeled Beans 5-75
How to Model Cross Component References 5-75
How to Display the Implementing Classes for Modeled Beans 5-76
How to Display the Source Code for a Modeled Bean 5-76
How to Change the Accessibility of a Property or Method 5-77
How to Reverse Engineer a Diagrammed JPA Entity 5-77
Modeling with Database Diagrams 5-77
Working with the Database Modeling Features 5-78
How to Create a Database Diagram 5-78
How to Create an Offline Database Object 5-79
How to Create a Foreign Key 5-79
How to Use Templates to Create Database Objects 5-79
How to Add and Create Private and Public Synonyms 5-79
How to Add and Create a Sequence 5-80
How to Add and Create Tables 5-80
How to Change the Database or Schema 5-81
How to Create Database Views and Add Database Objects 5-81
How to Define a Base Relation Usage 5-81
How to Create Join Objects 5-82

6 Versioning Applications with Source Control

About Versioning Applications with Source Control 6-1
Downloading Source Control Extensions in Oracle JDeveloper 6-1
Setting Up and Configuring Source Control 6-2
Setting Up Subversion and JDeveloper 6-2
Installing Subversion Client Software 6-2
Checking the Subversion Client Installation 6-3
Creating a Subversion Connection 6-3
Editing a Subversion Connection 6-4
Exporting Subversion Repository Connection Details 6-4
Importing Subversion Repository Connection Details 6-5
Connecting to a Subversion Repository Through a Proxy Server 6-5
Exporting Subversion Controlled Files from the Working Copy 6-6
Exporting Files from the Subversion Navigator 6-6
How to Set Up and Configure a Git Repository 6-7
How to Set Up Perforce with JDeveloper 6-7
Installing Perforce Components for use with JDeveloper 6-8
Perforce Client Installation 6-8
JDeveloper Installation 6-8
Configuring JDeveloper for Use with Perforce 6-9

Selecting Perforce as the Version System 6-9
How to Set Up Team System and JDeveloper 6-9
Setting Up Team System for Use with JDeveloper 6-10
Configuring JDeveloper for Use with Team System 6-10
Selecting Team System as the Versioning System 6-11
Setting the Team System Workspace to use JDeveloper 6-11
Creating a JDeveloper Project for the Workspace Files 6-11
Getting Versions of Files from the Team System Server 6-12
Adding Files to Team System Control 6-12
Versioning Applications With Mercurial 6-13
About Mercurial Visualization Features 6-13
How to Install Mercurial 6-13
How to Set the Path to the Mercurial Executable 6-14
How to Clone an External Mercurial Repository 6-14
How to Place Projects Under Version Control 6-15
How to Merge File Revisions 6-15
How to Commit Changes 6-16
Setting Up and Configuring a Source Repository 6-16
How to Create a Source Repository 6-17
Creating a Subversion Repository 6-17
Initializing a New Git Repository 6-18
Making a Local Copy of an Existing Git Repository 6-18
Adding New Files to an Existing Git Repository 6-18
Creating a Local CVS Repository 6-19
Importing JDeveloper Project Files Into CVS 6-19
Bringing Files Under Perforce Control 6-19
How to Connect to a Source Control Repository 6-21
Viewing Subversion Repository Content 6-21
Logging In to CVS 6-21
Accessing Local Files with CVS 6-22
Connecting to Perforce 6-23
Making Multiple Connections to Perforce 6-23
Configuring JDeveloper for the Source Repository 6-24
How to Configure CVS For Use with JDeveloper 6-24
Configuring for SSH Level 1 (SSH) 6-24
Configuring for SSH Level 2 (SSH2) 6-25
Editing and Watching Files in CVS 6-25
How to Load the Repository with Content 6-27
Importing JDeveloper Files Into Subversion 6-27
Importing a Project to Subversion 6-28
Adding a File to Subversion Automatically 6-28
Adding Files Individually to Subversion 6-29

Moving Files from Remote Repositories in Git 6-29
Importing JDeveloper Files Into Perforce 6-30
Updating a Project, Folder, or File in CVS 6-30
How to Create a WebDAV Connection 6-31
WebDAV Server Requirements 6-31
Creating a WebDAV Connection 6-32
Accessing a WebDAV-Enabled Server Via a Proxy Server 6-33
Modifying a WebDAV Connection 6-33
Refreshing a WebDAV Connection 6-33
Deleting a WebDAV Connection 6-34
Working with Files in Source Control 6-34
How to Check Out Files 6-34
Checking Out Files from the Subversion Repository 6-35
Checking Out Files in Git 6-36
Checking Out CVS Modules 6-36
Editing Files in Perforce 6-36
How to Update Files with Subversion 6-37
Updating Files from the Subversion Repository 6-37
Updating Individual Files in the Subversion Repository 6-37
Removing Files from Subversion Control 6-38
Working with Files in CVS 6-38
Refreshing the Display of CVS Objects 6-38
Refreshing the Status of Objects in JDeveloper 6-39
Adding and Removing Files in CVS 6-39
How to Work with New and Changed Files in Git 6-40
Adding a File to a Git Repository 6-41
Adding Multiple Files to a Git Repository 6-41
Create a Git Stash 6-41
Committing a Change to the Git Repository 6-41
Committing Multiple Files to the Git Repository 6-42
Applying a Git Stash 6-42
How to Work with Files in Perforce 6-43
Synchronizing Local Files With the Controlled Versions in Perforce 6-43
Synchronizing Files With the Perforce Navigator 6-43
Filtering Files By Perforce Workspace 6-45
Refreshing the Status of Files under Perforce Control 6-45
Deleting Files from Perforce 6-46
How to Lock and Unlock Files 6-46
Locking and Unlocking Files in CVS 6-46
Editing Files in Perforce 6-47
Checking Out Files in Team System 6-48
Viewing the Status of a File in Team System 6-48

Refreshing the Status of Files in Team System 6-48
How to Check In Changed Files 6-49
Committing Files to the Subversion Repository 6-49
Committing Changes to the Git Repository 6-50
Committing Changes to CVS 6-51
Submitting Changed Files to the Perforce Depot 6-52
How to Check In Files to Team System 6-52
How to Use Change Sets and Changelists 6-52
Using Change Sets in Subversion 6-53
Creating a Perforce Changelist 6-54
How to Annotate a Perforce Revision or Changelist 6-54
Adding Files to a Perforce Changelist 6-55
Submitting a Perforce Changelist 6-55
Using the Perforce Changelist Browser 6-55
How to Use Comment Templates for Checkins 6-56
Creating Templates 6-56
Sharing Templates 6-57
Selecting and Using Templates at Checkin 6-57
Shelving and Unshelving Team System Files 6-58
Deleting Team System Files 6-59
Working with Branches and Tags 6-59
How to Create Branches 6-59
Working with Branches and Tags in Subversion 6-60
Creating a New Branch in Git 6-60
Creating a New Branch in CVS 6-61
How to Use Branches 6-61
Checking Out a Branch in Git 6-62
Merging a Branch in Git 6-62
Using Branches in CVS 6-62
How to Create Tags 6-64
Creating Tags in Git 6-64
Creating and Assigning CVS Tags 6-65
Deleting CVS Tags 6-65
How to Use Tags 6-66
Using Tags in Git 6-66
Using Tags in CVS 6-66
How to Use Properties in Subversion 6-67
About Subversion Properties 6-67
Working with Subversion Properties 6-68
Viewing File and Property Status 6-69
Resolving Property Conflicts in Subversion 6-70
Working with File History, Status and Revisions 6-70

File History 6-70
Refreshing the Status of Files Under Subversion Control 6-71
Replacing a File with the Subversion Base Revision 6-71
How to Undo or Revert Changes 6-71
Reverting Files to their Previous State in Subversion 6-71
Reverting Changes to Files in Git 6-72
Working with Revisions and Tags in CVS 6-72
How to Merge Changes from Different Files 6-72
Comparing Files in Subversion 6-73
Resolving Conflicts in File Versions 6-73
Using the Merge Tool to Resolve Conflicts 6-74
Using the Subversion Merge Wizard 6-74
Working with File Versions and History in CVS 6-75
Merging Files in CVS 6-75
Comparing Files in CVS 6-76
Replacing a File with a CVS Revision 6-76
Viewing the History and Status of a File in CVS 6-77
Working with File Versions in Perforce 6-77
Working with File Versions in Team System 6-78
Resolving Conflicts in Team System File Versions 6-78
Undoing Changes to Team System Files 6-78
Replacing a File with the Team System Base Version 6-79
Viewing the History of a File 6-79
Comparing Files In Team System 6-79
Using an External Diff Tool with CVS 6-80
Integrating a Third Party Diff Utility 6-81
Integrating other CVS Commands 6-81
Working with Patches in Source Control 6-82
How to Create and Apply Patches 6-83
Creating Patches 6-83
Applying Patches 6-83

7 Getting Started with Developing Java Applications

About Developing Java Applications 7-1
Using the Java Source Editor 7-1
Using Code Insight 7-2
Using Code Insight to Add Annotations to Your Java Code 7-2
Using Code Peek 7-3
Using Scroll Tips 7-3
Using InfoTips 7-3
Searching Incrementally 7-4

Using Shortcut Keys 7-4
Bookmarking 7-4
Browsing Java Source 7-5
Using Code Templates 7-5
Setting Preferences for the Java Source Editor 7-5
How to Set Comment and Brace-Matching Options for the Java Source Editor 7-6
How to Set Import Statement Sorting Options for the Java Source Editor 7-6
How to Choose a Coding Style 7-7
How to Edit a Coding Style 7-7
How to Set Up a Coding Style Using an Extension 7-7
Using Toolbar Options 7-8
Using the Quick Outline Window 7-9
Working with the Java UI Visual Editor 7-10
Java Swing and AWT Components 7-11

8 Working with Java Code

About Working with Java Code 8-1
Navigating in Java Code 8-1
How to Browse Java Elements 8-2
Browsing a Java Element Directly in JDeveloper 8-2
How to Locate the Declaration of a Variable, Class, or Method 8-2
How to Find the Usages of a Class or Interface 8-3
How to Find the Usages of a Method 8-3
How to Find the Usages of a Field 8-4
How to Find the Usages of a Local Variable or Parameter 8-4
Identifying Overridden or Implemented Method Definitions 8-5
How to View the Hierarchy of a Class or Interface 8-5
Viewing the Hierarchy of a Class or Interface in the Java Source Editor 8-5
Stepping Through the Members of a Class 8-5
Editing Java Code 8-6
How to Create a New Java Class or Interface 8-6
How to Implement a Java Interface 8-7
How to Override Methods 8-7
How to Convert an Anonymous Inner Class to a Lambda Expression 8-8
How to Use Code Templates 8-8
Using Predefined Code Templates 8-9
How to Expand or Narrow Selected Text 8-14
How to Surround Code with Coding Constructs 8-15
How to Fold Code 8-15
Adding an Import Statement 8-16
How to Organize Import Statements 8-17

Using ojformat 8-17
Editing with the Java Visual Editor 8-18
Protecting Code 8-19
How to Add Documentation Comments 8-20
How to Update Documentation Comments 8-21
How to Set Javadoc Properties for a Project 8-21
How to Customize Documentation Comment Tags 8-21
How to View Javadoc for a Code Element Using Quick Javadoc 8-22
How to Preview Documentation Comments 8-23
How to Audit Documentation Comments 8-23
How to Build Javadoc 8-23
How to Create References to Missing Annotation Elements 8-24
Using the JOT Structure Window 8-24
How to Display Comments and Blank Lines in the JOT Structure Window 8-25
How to Set the Refresh Mode in the JOT Structure Window 8-25
Refactoring Java Projects 8-26
Refactoring on Java Class Diagrams 8-27
How to Invoke a Refactoring Operation 8-27
How to Preview a Refactoring Operation 8-28
How to Rename a Code Element 8-29
How to Delete a Code Element 8-30
Refactoring Classes and Interfaces 8-30
How to Duplicate a Class or Interface 8-31
How to Extract an Interface from a Class 8-32
How to Extract a Superclass 8-33
How to Use Supertypes Where Possible 8-33
How to Convert an Anonymous Class to an Inner Class 8-34
How to Move an Inner Class 8-34
Refactoring Class Members 8-35
Moving a Static Method 8-35
Moving a Non-static Method 8-35
How to Change a Method to a Static Method 8-36
How to Change the Signature of a Method 8-36
How to Pull Members Up into a Superclass 8-37
How to Push Members Down into Subclasses 8-38
How to Introduce a Field 8-38
How to Inline a Method Call 8-39
How to Introduce a Variable 8-39
How to Introduce a Parameter 8-40
How to Introduce a Constant 8-40
How to Extract a Method 8-41
How to Extract a Class 8-42

How to Replace a Constructor with a Factory Method 8-42
How to Encapsulate a Field 8-43
How to Invert a Boolean Expression 8-43
Refactoring XML Schemas 8-44

9 Building Java Projects

About Building Java Projects 9-1
Building with Make and Rebuild Commands 9-2
How to Set Compiler Preferences 9-2
Compiling with Make 9-2
Compiling with Rebuild 9-3
Understanding Dependency Checking 9-3
Compiling Applications and Projects 9-4
How to Configure Your Project for Compiling 9-4
How to Specify a Native Encoding for Compiling 9-5
Compiling from the Command Line 9-6
Cleaning Applications and Projects 9-6
How to Clean 9-7
Cleaning a Project 9-7
Cleaning an Application 9-7
Cleaning and Refreshing an Application 9-7
Building with Apache Ant 9-8
Create an Ant Build File at Application Level 9-8
Create an Ant Build File at Project Level 9-8
Create an Empty Ant Build File 9-9
Running Ant on Project Buildfile Targets 9-9
Using the Ant Tool in the IDE 9-10
Building and Running with Apache Maven 9-10
Understanding Repositories 9-11
Understanding Maven Plugins 9-11
Understanding Dependencies 9-12
Understanding the Project Object Model 9-12
Understanding the Settings File 9-13
Selecting the POM File 9-13
Installing Maven 9-14
Before You Begin 9-14
How to Create Maven POM Files 9-15
Using the Context Menu in the POM file 9-16
How to Specify and Manage Remote Repositories 9-17
Populating the Repository 9-18
Synchronizing POM and Project Files 9-19

Customizing Maven Synchronization 9-20
How to Match the Default Maven Structure When You Create an Application 9-21
How to Create Maven Projects Using Maven Archetypes 9-21
What Happens When You Create a New Maven Application 9-23
How to Run Maven Goals on POM Files 9-23
How to Create a Maven POM for a Project 9-24
Auditing Maven Applications 9-25
Configuring Test Settings 9-25
Understanding Code Insight 9-25
Using the WebLogic Maven Plugin in JDeveloper 9-26
Using ojdeploy and ojmake 9-26
Understanding Continuous Delivery and Continuous Integration 9-27

10 Testing and Profiling Java Application Projects

About Testing and Profiling Java Application Projects 10-1
Testing Java Application Projects with Unit Tests 10-2
Creating a JUnit Test for a Java Project 10-2
How to Create a JUnit Custom Test Fixture 10-2
How to Create a JUnit JDBC Test Fixture 10-3
Creating a JUnit Test Case 10-3
How to Add a Test to a JUnit Test Case 10-4
Creating a JUnit Test Suite 10-5
How to Create a Business Components Test Suite 10-5
How to Create a Business Components Test Fixture 10-7
How to Update a Test Suite with all Test Cases in the Project 10-8
How to Run JUnit Test Suites 10-8
Starting a Profiling Session 10-8
Starting and Profiling JDeveloper Applications Simultaneously 10-8
How to Calibrate the Profiler 10-9
Attaching the Profiler to a Running JDeveloper Applications 10-9
Profiling External Applications 10-10
Taking and Accessing Snapshots of Profiling Data 10-10
Taking Snapshots at the End of a Profiling Session 10-10
Taking Snapshots During a Profiling Session 10-11
Starting and Stopping the Application Finished Dialog 10-12
Accessing Snapshots 10-12
Taking a Heap Dump 10-13
Viewing UI Elements with Heap Walker 10-14
Image Preview Use Cases 10-15
How to Analyze a Heap Dump Using Object Query Language (OQL) 10-16
OQL Examples 10-16

OQL built-in objects and functions 10-17
Selecting Multiple Values 10-21
Other Examples 10-24
Setting a Profiling Point 10-25
Profiling Telemetry 10-26
Profiling Methods 10-26
Profiling Specific Methods 10-27
Profiling Objects 10-27
Profiling Specific Objects 10-28
Profiling Threads 10-29
Profiling Locks 10-30
Additional Functions when Running a Profiling Session 10-30

11 Auditing and Monitoring Java Projects

About Auditing and Monitoring Java Projects 11-1
Auditing Java Projects 11-1
Understanding Audit Rules 11-2
Understanding Audit Metrics 11-2
Using the Auditing Tools 11-3
Using the Audit Window Report Panel 11-3
Using the Audit Window Toolbar 11-3
Using the Audit Window Context Menu 11-4
How to Audit Java Code in JDeveloper 11-5
Auditing Java Code from the Command Line 11-6
Working with Audit Profile 11-8
Create an Audit Profiler 11-9
Sealing a Profile 11-10
Disabling Suppression Schemes 11-10
How to Delete an Audit Profile 11-11
How to Import or Export an Audit Profile 11-11
How to Run Audit to Generate an Audit Report 11-11
How to Audit Unserializable Fields 11-12
How to Audit Serializable Fields That Do Not Have serialVersionUID 11-12
Viewing an Audit Report 11-13
How to Organize Audit Report Rows 11-14
Using Filters with Reports 11-14
How to Filter Audit Report Rows 11-15
How to Save an Audit Report 11-15
How to Fix an Audit Rule Violation 11-16
How to Fix a Construct's Audit Rule Violations 11-16
How to Hide Audit Rule Violations 11-17

How to Hide Audit Report Measurements 11-17
Monitoring HTTP Using the HTTP Analyzer 11-17
How to Use the Log Window 11-18
How to Use the Test Window 11-19
How to Use the Instances Window 11-21
What Happens When You Run the HTTP Analyzer 11-22
How to Specify HTTP Analyzer Settings 11-22
How to Use Multiple Instances 11-22
How to Configure External Web Browsers 11-23
Using SSL with the HTTP Analyzer 11-24
How to Use SSL with the HTTP Analyzer 11-25
How to Run the HTTP Analyzer 11-26
How to Debug Web Pages Using the HTTP Analyzer 11-26
How to Edit and Resend HTTP Requests 11-27
How to Use Rules to Determine Behavior 11-27
Using the Pass Through Rule 11-27
Using the Forward Rule 11-28
Using the URL Substitution Rule 11-28
Using the Tape Rule 11-28
How to Set Rules 11-29
Using the HTTP Analyzer with Web Services 11-29
Testing Web Services with the HTTP Analyzer 11-30
Using the HTTP Analyzer with RESTful Web Services 11-30
Using the HTTP Analyzer with WebSockets 11-33
Using the HTTP Analyze with Fast Infoset 11-33
Reference: Troubleshooting the HTTP Analyzer 11-33
Running the HTTP Analyzer While Another Application is Running 11-33
Changing Proxy Settings 11-34

12 Running and Debugging Java Projects

About Running and Debugging Java Programs 12-1
Understanding the Processes Window 12-2
Configuring a Project for Running 12-2
How to Choose a Run Configuration 12-2
How to Create a Run Configuration 12-3
How to Run a Project or File 12-3
How to Run a Project from the Command Line 12-4
How to Change the Java Virtual Machine 12-4
Macros 12-4
Setting the Classpath for Programs 12-5
Setting the CLASSPATH Environment Variable (for java.exe) 12-5

Using the JDeveloper Library CLASSPATH 12-6
Setting the CLASSPATH to Include Your Projects 12-6
Setting the CLASSPATH Parameter (for java.exe) 12-6
Debugging Java Programs 12-7
Understanding the Debugger Icons 12-8
Debugging an Application Deployed to Integrated WebLogic Server 12-10
How to Debug a Project in JDeveloper 12-11
How to Edit and Recompile 12-12
Using FastSwap Deployment to Minimize Redeployment 12-12
How FastSwap Deployment Works 12-12
Supported FastSwap Application Configurations 12-13
Enable FastSwap in Your Application 12-13
Overview of the FastSwap Process 12-13
Application Types and Changes Supported with FastSwap 12-14
Limitations When Using FastSwap 12-16
Handling Unsupported FastSwap Changes 12-16
How to Debug ADF Components 12-17
How to Use JDeveloper Debugger to Execute a Program 12-17
How to Configure a Project for Debugging 12-18
How to Set the Debugger Start Options 12-18
How to Launch the Debugger 12-19
How to Export Debug Information to a File 12-20
Using the Source Editor When Debugging 12-20
Using Context Menu Items 12-21
Using Tooltips 12-21
Using Java Expressions in the Debugger 12-21
Moving Through Code While Debugging 12-22
How to Step Into a Method 12-23
How to Step Over a Method 12-24
Controlling Which Classes Are Traced Into 12-24
How to Step Through Behavior as Guided by Tracing Lists 12-25
How to Locate the Execution Point for a Thread 12-25
How to Run to the Cursor Location 12-26
How to Pause and Resume the Debugger 12-26
How to Terminate a Debugging Session 12-27
How to View the Debugger Log 12-27
Using the Debugger Windows 12-27
How to Open Debugger Windows 12-28
How to Use the Breakpoints Window 12-28
How to Use the Data Window 12-28
How to View Array Elements in the Data Window 12-29
How to Use the Smart Data Window 12-29

How to Use the Watches Window 12-29
How to Add a Watch from the Source Editor 12-30
How to Watch a Static Field 12-30
How to Edit a Watch 12-30
How to Delete a Watch 12-31
How to Use the Inspector Window 12-31
How to Use the Heap Window 12-32
Using the Stack Window 12-32
How to Use the Stack Window 12-32
How to View the Stack of a Tread 12-32
How to Use the Classes Window 12-33
How to Change the View Order 12-33
How to Use the Monitors Window 12-33
How to Use the Threads Window 12-34
How to Set Preferences for the Debugger Windows 12-34
How to Specify Which Columns Display in the Window 12-35
Managing Breakpoints 12-35
Understanding Verified and Unverified Breakpoints 12-36
Understanding Deadlocks 12-36
Understanding the Deadlock Breakpoint 12-37
Understanding Grouped Breakpoints 12-38
How to Edit Breakpoint Options 12-38
Editing a Breakpoint 12-39
How to Set Source Breakpoints 12-39
How to Control Breakpoint Behavior 12-40
How to Delete a Breakpoint 12-40
How to Disable a Breakpoint 12-41
How to Enable a Breakpoint 12-41
How to Set Instance Breakpoints 12-41
How to Set Exception Breakpoints 12-42
How to Make a Breakpoint Conditional 12-43
Using Pass Count Breakpoints 12-43
How to Examine Breakpoints with the Breakpoints Window 12-44
How to Manage Breakpoint Groups 12-44
How to Create a Breakpoint Group 12-44
How to Move a Breakpoint into a Breakpoint Group 12-45
Enabling Disabling or Removing a Breakpoint Group 12-45
Examining Program State in Debugger Windows 12-45
How to Inspect and Modify Data Elements 12-45
How to Inspect a Data Item 12-46
How to Modify the Value of a Variable 12-46
How to Modify Expressions in the Inspector Window 12-47

How to Show and Hide Fields in the Filtered Classes List 12-47
Debugging Remote Java Programs 12-48
How to Start a Java Process in Debug Mode 12-48
How to Use a Project Configured for Remote Debugging 12-49
How to Configure JPDA Remote Debugging 12-50

13 Implementing Java Swing User Interfaces

About Applications Developed in Earlier Versions 13-1
About Java Swing UI Components and Containers 13-2
Designing Java GUIs 13-2
About Guarded Blocks 13-3
How to Create a Form 13-3
Understanding the Forms You Can Create 13-4
Adding Components 13-4
How to Set Component Properties 13-5
How to Select Components in Your User Interface 13-6
How to Select a Single Components 13-6
How to Select Multiple Components 13-6
How to Align Components 13-6
How to Size Components 13-7
Working with Containers 13-8
Reordering Components Within a Container 13-8
Working with Layout Managers 13-9
How to Set the Layout Manager 13-10
Understanding FreeDesign Layout 13-10
How to Set Layout Properties 13-11
Understanding Layouts Provided with JDeveloper 13-11
Using BorderLayout 13-13
Using CardLayout 13-14
Using FlowLayout 13-14
Changing the Alignment 13-15
Changing the Gap 13-15
Changing the Order of Components 13-15
Using GridBagLayout 13-15
Adding Components to a GridBagLayout Container 13-16
How to Set GridBagConstraints in the Properties Window 13-16
How to Use the GridBag Customizer 13-17
Using GridLayout 13-19
Previewing a User Interface 13-20
How to Create Accessible Forms 13-20
Working with Event Handling 13-20

How to Attach Event Handling Code to Menu Events 13-21
How to Attach Event-Handling Code to a Component Event 13-21
How to Quickly Create an Event Handler for a Component's Default Event 13-22
How to Modify GUI Source Code 13-22
Modifying GUI Form Code Outside of the IDE 13-23
How to Modify Code Generation for a Property 13-23
Working with the UI Debugger 13-24
Working with UI Debugger Windows 13-24
How to Start the UI Debugger 13-25
Examining the Application Component Hierarchy 13-25
How to Display Component Information in the Watches Window 13-26
How to Inspect a UI Component in an Properties window 13-26
How to Trace Events Generated by Components 13-27
How to Show Event Listeners 13-27
Remote Debugging GUI Applications 13-28
How to Remote Debug GUI Applications 13-28
How to start the JDeveloper IDE for Remote UI Debugging 13-29
Automatic Discovery of Listeners 13-29

14 Working with JavaBeans

About Working with JavaBeans 14-1
Using JavaBeans in JDeveloper 14-1
How to Implement an Event-Handling Method 14-2
What Happens When You Create an Event-Handling Method 14-3
Understanding Standard Event Adapters 14-3
How to Create an Event Set 14-3
How to Make a Component Capable of Firing Events 14-4

15 Getting Started with Developing Java EE Applications

About Developing Java EE Applications 15-1
Java EE and Oracle Application Developer Framework 15-1
Using Web Page Tools 15-2
Using Enterprise JavaBeans and Java Persistence Components 15-2
Using Oracle TopLink 15-2
Understanding Secure Applications 15-3
Working With Applications That Use XML 15-3
Working With Applications That Use Web Services 15-3

16 Developing Applications Using Web Page Tools

About Developing Applications Using Web Page Tools 16-1
- Using the Source Editor 16-1
- Source Editor Features 16-2
- Working in the Visual Editing Environment 16-3
  - Primary Visual Editing Features 16-4
  - Additional Editing Tools and Features 16-4
  - Keystroke Commands 16-5
  - How to Expand and Collapse Container Elements 16-7
  - How to Change the Default Environment Settings 16-7
  - How to Display Invisible Elements 16-7
  - How to Execute JSP Tags 16-8
  - How to Display JSP Tags by Name Only 16-8
  - How to Change Keyboard Preferences 16-8
  - How to Select an Element 16-9
  - How to Select Multiple Elements 16-9
  - How to Select a Range of Elements 16-10
  - How to Select Text 16-10
  - How to Select Insertion Points in the Visual Editor 16-10
  - How to Select Insertion Points in the Structure Window 16-11
  - How to Insert Elements from the Components Window 16-11
  - How to Set and Modify Element Properties 16-12
  - How to Set a Data Source for a Property 16-12
  - How to Set Properties for Multiple Elements 16-13
  - How to Cut Page Elements 16-13
  - How to Delete Page Elements 16-14
  - How to Copy Page Elements 16-14
  - How to Move Page Elements 16-14
  - How to Resize Page Elements 16-14
  - How to Create and Edit a Data Table 16-15
  - How to Work with Data Table Columns 16-15
  - How to Work with Panel Grids 16-16
  - How to Paste Markup Code 16-16
  - How to View and Edit Web Page Head Content 16-17
- Using the Properties Window 16-17
  - How to Edit Properties 16-19
  - How to Write Custom Property Editors 16-19
- Using the Components Window 16-19
  - Using the Components Window 16-20
- Using the Overview Editor for JSF Configuration Files 16-21
  - How to Add, Delete, or Edit JSF Configuration Elements 16-21

How to Work with JSF Configuration Child Elements 16-22
Planning Your Page Flows With JSF Navigation Diagrams 16-22
How to View Your Navigation Diagrams 16-22
How to Add an Element to a JSF Navigation Diagram 16-23
How to Add a JSF Navigation Case to a JSF Navigation Diagram 16-24
How to Add a Note to a Navigation Diagram 16-25
How to Attach Notes to Elements in a Navigation Diagram 16-25
How to Set Layout Default Styles on a Navigation Diagram 16-26
How to Refresh Your Navigation Diagram to Reflect Changes in the Configuration File. 16-26
How to Use the Navigation Diagrammer to Manage JSF Pages 16-27
Editing and Deleting Navigation Cases 16-27
How to View Navigation Case Properties 16-28
How to Publish a Diagram as a Graphic 16-28
How to Use Code Insight For Faster Web Page Coding 16-28
Developing Applications with JavaServer Faces 16-30
Building Your JSF Application 16-30
How to Build Your Application Framework 16-31
How to Create Your JSF Pages and Related Business Services 16-31
How to Choose a Business Services 16-31
Building your JSF Business Component Framework 16-32
Using the JSF Tag Libraries 16-33
Using Standard JSF Component Tag Attributes 16-39
How to Create Managed Beans 16-39
Creating Managed Beans 16-40
How to Create Managed Beans with the Overview Editor 16-41
How to Add Methods to a Managed Bean 16-41
How to Bind Components with EL Expressions 16-41
Constructing an EL Expression 16-43
How to Create Composite Components 16-43
Using Automatic Component Binding 16-43
Turning the Automatic Bind Option On and Off 16-45
How to Set a Page to Auto Bind to Managed Beans 16-46
How to Value Bind a Component to a Property 16-46
How to Manually Bind Component Instances to Properties 16-46
How to Bind an Existing Method with Auto Component Binding 16-47
How to Bind a New Default Method with Auto Binding On 16-47
How to Bind a New Default Method with Auto Binding Off 16-47
Using Localized Resource Bundles in JSF 16-48
How to Use Localized Resource Bundles in JSF 16-48
How to Work with Facets 16-49
How to Build JSF Views with Facelets 16-50

Creating a Facelet 16-51
How to Register a Standard Converter Using a Supplied Tag 16-51
How to Register a Standard Converter That Does Not Have Tag 16-52
How to Register a Standard Validator Using a Standard Tag 16-52
How to Display a Message Next to the Component that Generated the Conversion or Validation Errors 16-53
How to Register a Custom Converter or Validator in the JSF Application Configuration 16-53
How to Edit a Custom Converter or Validator in a Configuration File 16-54
How to Delete a Custom Converter or Validator in a Configuration File 16-54
How to Register a Custom Converter on a Component Using a Converter Identifier 16-54
How to Register a Custom Converter on a Component Using a Value Binding Expression 16-55
How to Register a Custom Validator Instance on a Component 16-55
How to Bind a Component to a New Validator Method 16-55
Using the Standard Converter and Validator Tags and Syntax 16-56
How to Display Error Messages 16-58
How to Display All Error Messages Generated in a Page 16-58
How to Replace the Standard Message Texts in JSF 16-59
How to Add Information About a Form Field to Which a Message Refers 16-59
How to Change the Appearance of Error Messages a JSF Page 16-60
Configuring JSF Applications 16-61
How to Use the Overview Editor to set the <application> Element 16-61
How to Add a Bean to a JSF Configuration File 16-61
How to Create a New JSF Configuration File 16-62
How to Edit a JSF Configuration File 16-62
Converting a Project to Facelets 16-63
Things to Consider Before Converting 16-63
Previewing your Conversion Status in the Log Windows 16-64
How to Convert your Project to Facelets 16-64
Running and Testing JSF Applications 16-65
How to Run and Test Individual Packages 16-65
How to Run and Test an Entire Project 16-65
Developing Applications with HTML Pages 16-66
Building Your HTML Pages 16-66
How to Create an HTML Page 16-66
Using the HTML Core Components 16-67
How to Save JSP Files as HTML 16-68
Working with HTML Text 16-69
How to Add Text to an HTML Page 16-69
How to Set Text Properties 16-69
Working with HTML Images 16-69
How to Insert an Image on a HTML File 16-70

How to Delete an Image From an HTML File 16-70
How to Resize an Image in an HTML File 16-70
How to Use an Image as an HTML Background 16-71
How to Move an Image in an HTML File By Dragging 16-71
Working with HTML Tables 16-72
How to Add Text to a Table Cell 16-72
How to Remove Content from One or More Selected Cells 16-72
How to Format Tables and Cells 16-72
How to Set Table and Cell Properties 16-73
How to Set Table and Cell Properties Using the Visual Editor Toolbar 16-73
How to Resize a Table 16-73
How to Change the Size of Rows and Columns 16-74
How to Add Rows or Columns to a Table 16-74
How to Remove Rows or Columns in a Table 16-74
How to Merge Table Cells 16-75
How to Split a Table Cell 16-75
How to Change the Display Order in a Table Structure 16-76
How to Change the Display Order of Rows Using the Structure Window 16-76
How to Increase Row or Column Span in a Table 16-76
How to Reduce Row or Column Span in a Table 16-77
Working with HTML Forms 16-77
How to Create an HTML Form 16-78
How to Delete an HTML Form Element 16-78
How to Insert an HTML Form Field or Button 16-79
How to Change the Form Method from the Context Menu 16-79
How to Set Form Processing Information Using the Properties window 16-79
How to Delete a Form Field or Button 16-80
Working with Cascading Style Sheets 16-80
Selecting and Grouping CSS Elements 16-81
Working with Grouped Elements 16-82
How to Create a Simple Cascading Style Sheet 16-83
How to Set or Modify CSS Selector Properties and Values 16-83
How to Format Text with CSS Properties 16-84
How to Edit a CSS File 16-84
Working with Java Server Pages 16-85
Building Your JSP Application 16-85
JSP Core Components 16-85
How to Create JSP Pages 16-87
How to Register a Servlet Filter in a JSP Page 16-87
Understanding Flow Control in JSPs 16-87
How to Handle JSP Flow Control 16-88
Debugging and Deploying JSPs 16-88

How to Debug a JSP 16-89
How to Create a Web Deployment Descriptor 16-89
How to Edit Web Deployment Descriptor Properties 16-90
How to Edit a Web Deployment Descriptor as an XML File 16-90
Running a JSP 16-90
How to Run a JSP 16-90
Dynamically Modifying JSP Files While They are Running 16-91
Running JSPs with ADF Business Components Application Modules 16-91
Working with Timestamps on Source JSPs 16-91
Understanding JSP Segments 16-91
Developing Applications with Java Servlets 16-92
Understanding Servlet Support in JDeveloper 16-92
What You May Need to Know About Servlet Filters 16-92
What You May Need to Know About Servlet Listeners 16-93
How to Generate an HTTP Servlet 16-93
Implementing Basic Methods for an HTTP Servlet 16-94
How to Use the HTTPServletRequest Object 16-94
How to Use the HTTPServletResponse Object 16-95
How to Create a Servlet Filter 16-96
How to Create a Servlet Listener 16-96
Registering a Servlet Filter in a JSP Page 16-97
How to Run a Servlet 16-98
How to Debug a Servlet 16-99
How to Deploy a Servlet 16-99
Developing Applications with Script Languages 16-99
How to Work with JavaScript Code Insight 16-100
How to Use Breadcrumb Support 16-101
Working with Script Languages 16-101
How to Create a Script 16-101
How to Add Script Language Elements to an HTML or JSP Page 16-102
How to Set Syntax Highlighting 16-103
How to Associate JavaScript File Extensions 16-103
How to Create a JSON File 16-104
How to Use Structure Pane Support 16-104
Refactoring JavaScript Code 16-105
Finding Usages of Code Elements 16-105
Renaming a JavaScript Code Element 16-106
Deleting a JavaScript Code Element 16-106
How to Preview a Refactoring Operation 16-107
How to Reformat JavaScript Code 16-108
How to Change Code Formatting Preferences 16-108
How to Use Code Folding 16-108

How to Refactor and Move a File 16-109
Working with JSP and Facelet Tag Libraries 16-109
Using Tag Libraries with Your Web Pages 16-109
How To Add, Delete or Edit Project Level Tag Libraries 16-110
How to Browse to a JSP Tag Library Descriptor (TLD) File 16-111
How to Browse Pages or Individual JSP Tags 16-111
How to Work with Custom Tag Libraries 16-111
How to Create a Custom JSP or Facelets Tag Library 16-111
How to Add your Custom Tag Library to the Components Window 16-112
How to Register a JSP or Facelet Tag Library 16-113
How to Edit a TLD File in the XML Source Editor 16-113
How to Add a Scripting Variable to a Tag 16-114
How to Deploy Your Custom JSP/Facelets Tag Library as a JAR File 16-114

17 Developing with EJB and JPA Components

About Developing with EJB and JPA Components 17-1
Support For EJB Versions and Features 17-1
Building EJB 3.x Applications and Development Process 17-4
EJB 3.x Application Development Process 17-5
Creating Entities 17-5
Creating Session Beans and Facades 17-5
Deploying EJBs 17-6
Testing EJBs Remotely 17-6
Registering Business Services with Oracle ADF Data Controls 17-6
How to Work with an EJB Business Services Layer 17-6
Using Java EE Design Patterns in Oracle JDeveloper 17-7
Using Java EE Contexts and Dependency Injection (CDI) 17-8
beans.xml File 17-9
Interceptor Binding Type 17-9
Qualifier Type 17-10
Scope Type 17-11
Stereotype 17-11
Building a Persistence Tier 17-12
About JPA Entities and the Java Persistence API 17-12
JPA Entities are POJOs 17-12
Metadata Annotations for O-R Mapping 17-13
Inheritance and Polymorphism Support 17-13
Simplified EntityManager API for CRUD Operations 17-14
Query Enhancements 17-14
How to Create JPA Entities 17-14
Using the Serializable Interface 17-15

About SDO For EJB/JPA 17-16
Using an EJB/POJO-based ADF-BC Service for Deployment to the SOA Platform 17-16
How to Create an SDO Service Interface for JPA Entities 17-16
How to Configure an EJB/POJO-based ADF-BC Service for Deployment to the SOA Platform 17-17
File Types Created to Support Your SDO Architecture 17-17
How to Generate Database Tables from JPA Entities 17-18
Annotations for EJB/JPA 17-18
EJB 3.x 17-19
JPA 2.0 17-19
How to Annotate Java Classes 17-19
Representing Relationships Between Entities 17-20
Java Persistence Query Language 17-21
JPA Object-Relational Mappings 17-21
How to Use Java Service Facades 17-21
How to Define a Primary Key for an Entity 17-22
Implementing Business Processes in Session Beans 17-23
Using Session Facades 17-24
How to Create a Session Bean 17-25
How to Create Session Beans in EJB Modules 17-26
How to Create Message-Drive Beans in EJB Modules 17-26
How to Add, Delete, and Edit EJB Methods 17-27
How to Add a Field to an EJB 17-27
How to Remove a Field From an EJB 17-27
Customizing Business Logic with EJB Environment Entries 17-28
Exposing Data to Clients 17-28
How to Identify Resource References 17-28
How to Specify a Primary Key for ADF Binding 17-28
How to Use ADF Data Controls for EJBs 17-29
Modeling EJB/JPA Components on a Diagram 17-29
Deploying EJBs as Part of an Web Application 17-29
Deploying EJB Modules and JPA Persistence Units 17-29
Deploying JPA Entity Beans 17-29
About EJB Modules 17-30
About JPA Persistence Units 17-30
How to Create a JPA Persistence Unit 17-30
How to Remove EJBs in a Module 17-31
How to Import EJBs into JDeveloper 17-31
Running and Testing EJB/JPA Components 17-31
How to Test EJB/JPA Components Using the Integrated Server 17-32
How to Test EJB/JPA Components Using a Remote Server 17-32

How to Test EJB Unit with JUnit 17-33

18 Developing Persistence in Applications Using Oracle TopLink

About Developing Persistence in Applications Using TopLink 18-1
Developing TopLink JPA Projects 18-2
How to Specify the JPA Version 18-2
How to Create Entities 18-3
How to Create and Configure a JPA Persistence Descriptor (persistence.xml) 18-3
How to Create Persistence Units 18-4
How to Configure Persistence Units 18-5
About Using JPA Mappings 18-6
Using Metadata Annotations 18-7
Using XML 18-7
Defaulting Properties 18-7
Configuring an Entity 18-7
Declaring Basic Property Mappings 18-7
Mapping Relationships 18-7
Mapping Inheritance 18-8
Mapping Embedded Objects 18-8
How to Use JPA Mappings 18-8
How to Create JPA Mapping Descriptors 18-9
How to Configure Persistence Unit Defaults 18-10
How to Set Access Type Defaults and Overrides 18-10
How to Generate Unique IDs for Primary Keys 18-11
How to Configure Queries 18-11
How to Specify Derived Identifiers in Mappings 18-12
Using TopLink Extensions 18-12
Developing Native TopLink Mappings 18-12
Designing Native TopLink Applications 18-12
Using Native TopLink in Application Design 18-13
Creating Native TopLink Metadata 18-13
Creating Project Metadata 18-14
Creating Session Metadata 18-14
Using Native TopLink Descriptors 18-14
Relational Descriptors 18-15
EIS Descriptors 18-15
XML Descriptors 18-15
Using Native TopLink Mappings 18-15
Relational Mapping Types 18-15
EIS Mapping Types 18-16
XML Mapping Types 18-17

Understanding the TopLink Editor 18-17
Managing TopLink Maps 18-17
Managing TopLink Sessions 18-18
Managing Persistence Configurations 18-18
The TopLink Structure View Toolbar 18-18
TopLink Project Elements in the Applications Window 18-19
TopLink Editor Tabs in the Editor Window 18-19
TopLink Project Elements in the Structure View 18-20
Using the TopLink Structure View Toolbar 18-20
TopLink Mapping Status Report in Message Log 18-20
Configuring TopLink Preferences 18-20
How to Create a Native TopLink Mapping Project 18-20
How to Use Converter Mappings 18-21
How to Automap TopLink Descriptors 18-21
Data Source Login Information 18-22
Developing Native TopLink Relational Projects 18-22
How to Create Relational Projects and Object Maps 18-22
How to Create Relational Descriptors 18-22
How to Configure Relational Descriptors 18-23
Developing Native TopLink XML Projects 18-24
How to Create XML Projects and Object Maps 18-24
How to Create XML Descriptors 18-25
How to Add XML Schemas 18-25
Developing Native TopLink EIS Projects 18-25
How to Create EIS Projects and Object Maps 18-26
How to Create EIS Descriptors 18-26
Using EIS Data Sources 18-26
Developing Native TopLink Sessions 18-27
How to Create a New Native TopLink Sessions Configuration File 18-27
How to Create Native TopLink Sessions 18-28
Acquiring Sessions at Runtime 18-28
How to Create Session Brokers 18-28
How to Create Data Source Logins 18-29
How to Create Connection Pools 18-29
Developing Native TopLink Applications 18-30
Using TopLink the Cache 18-30
Object Identity 18-31
Querying and the Cache 18-31
Handling Stale Data 18-31
Explicit Query Refreshes 18-31
Cache Invalidation 18-31
Cache Coordination 18-31

Cache Isolation 18-32
Cache Locking and Transaction Isolation 18-32
How to Configure the TopLink Cache 18-32
Using Queries 18-32
TopLink Query Languages 18-33
TopLink Query Types 18-33
How to Create Queries 18-33
Using Basic Query API 18-34
Using Advanced Query API 18-34
Redirect Queries 18-35
Historical Queries 18-35
Fetch Groups 18-35
Read-Only Queries 18-35
Interfaces 18-35
Inheritance Hierarchy 18-35
Additional Join Expressions 18-36
EJB Finders 18-36
Cursor and Stream Query Results 18-36
How to Create TopLink Expressions 18-36
Understanding TopLink Transactions 18-37
TopLink Transactions and the Unit of Work 18-37

19 Developing Secure Applications

About Developing Secure Applications 19-1
Understanding Java EE Applications and Oracle Platform Security Services for Java (OPSS) 19-1
Understanding Fusion Web Applications and ADF Security 19-1
Understanding Container-managed Security 19-1
Additional Functionality 19-2
Securing Applications in Phases 19-2
About Web Application Security and JDeveloper Support 19-3
Handling User Authentication in Web Applications 19-3
About Authentication Type Choices 19-3
BASIC authentication 19-4
FORM authentication 19-4
CLIENT-CERT authentication 19-4
Encrypting Passwords for a Target Domain 19-4
weblogic.security.Encrypt 19-5
How to Create an Identity Store 19-6
How to Add Test Users to the Identity Store 19-7
Managing Enterprise Roles in the Identity Store 19-7

How to Add Roles to the Identity Store 19-8
How to Manage Users Assigned to User Roles 19-8
How to View Assigned Enterprise Roles 19-8
How to Create a Credential Store 19-8
How to Add a Login Module 19-9
How to Authenticate Through a Custom Login Module 19-10
How to Add a Key Store 19-11
How to Enable an Anonymous Provider 19-11
How to Add Credentials to Users in the Identity Store 19-12
How to Choose the Authentication Type for the Web Application 19-12
Securing Application Resources in Web Applications 19-12
How to Secure Application Resources Using the jazn-data.xml Overview Editor 19-13
How to Secure ADF Resources Using ADF Security in Fusion Web Applications 19-13
Configuring an Application-Level Policy Store 19-14
How to Add Application Roles to an Application Policy Store 19-15
How to Add Member Users or Enterprise Roles to an Application Role 19-15
How to Create Custom Resource Types 19-16
How to Add Resource Grants to the Application Policy Store 19-16
How to Add Entitlement Grants to the Application Policy Store 19-17
How to Create a Custom JAAS Permission Class 19-17
How to Add Grants to the System Policy Store 19-18
Migrating the Policy Stores 19-18
How to Migrate the Policy Stores 19-18
Migrating Application Policies 19-19
Migrating Credentials 19-19
Migrating Users and Groups 19-20
Securing Development with JDBC 19-20

20 Developing Applications Using XML

About Developing Applications Using XML 20-1
Using the XML File Editors 20-1
Understanding XML Editing Features 20-2
Understanding the XML Editor Toolbar 20-3
How to Set Editing Options for the XML Editor 20-3
Working with XML Schemas 20-4
Working with Attributes in the XSD Visual Editor 20-4
What Happens When You Create an XML Schema in the XSD Visual Editor 20-5
Selecting XSD Components 20-5
Choice Component 20-6
All Component 20-6
Sequence Component 20-6

Cardinality and Ordinality 20-7
ComplexType Component 20-8
Attribute Group Component 20-8
Union Component 20-9
List Component 20-9
Working with XML Schema Substitution Groups 20-9
How to Import and Register XML Schemas 20-10
How to Generate Java Classes from XML Schemas with JAXB 20-10
Working with XSD Documents and Components 20-11
How to Display a Schema in Both Editors 20-11
How to Create an Image of the XSD Visual Editor Design Tab 20-11
How to Navigate with Grab Scroll in the XSD Visual Editor 20-12
How to Expand and Collapse the XSD Component Display 20-12
How to Zoom In and Out in the XSD Visual Editor 20-12
How to Select XSD Components 20-13
Escalating a Single Component 20-13
Selecting Multiple Components 20-13
What Happens When You Select a Component in the XSD Visual Editor 20-14
How to Select Target Positions for XSD Components 20-14
How to Insert XSD Components 20-15
Inserting XSD Components Using the Components Window 20-15
Inserting XSD Components Using the Context Menu 20-16
How to Cut XSD Components 20-16
How to Copy XSD Components 20-16
How to Delete XSD Components 20-17
How to Paste XSD Elements 20-17
How to Move XSD Components 20-17
Moving Components by Dragging 20-17
Moving Components by Cutting and Pasting 20-18
How to Set and Modify XSD Component Properties 20-18
How to Set Properties for Multiple XSD Components 20-19
Localizing with XML 20-19
What You May Need to Know About XLIFF Files 20-20
Developing Databound XML Pages with XSQL Servlet 20-20
Supporting XSQL Servlet Clients 20-20
How Can You Use XSQL Servlet? 20-21
How to Create an XSQL File 20-21
How to Edit XML Files with XSQL Tags 20-22
How to Check the Syntax in XSQL Files 20-22
How to Create XSQL Servlet Clients that Access the Database 20-22
Creating XSQL Servlet Clients for Business Components 20-23
What You May Need to Know About XSQL Error JBO-27122 20-24

How to Create a Custom Action Handler for XSQL 20-25
How to Deploy XSQL Servlets 20-26
How to View Output from Running XSQL Files as Raw XML Data 20-28
How to Create an XSL Style Sheet for XSQL Files 20-28
How to Format XML Data with a Style Sheet 20-29
How to Modify the XSQL Configuration File 20-30
Using XML Metadata Properties in XSQL Files 20-30
Using XML_ELEMENT 20-31
Using XML_ROW_ELEMENT 20-31
Using XML_CDATA 20-32
Using XML_EXPLICIT_NULL 20-32

21 Developing and Securing Web Services

About Developing and Securing Web Services 21-1
Developing Java EE Web Services Using JDeveloper 21-2
Securing Java EE Web Services Using JDeveloper 21-3
Discovering and Using Web Services 21-4
Using JDeveloper to Create and Use Web Services 21-4
How to Use Proxy Settings and JDeveloper 21-5
Using the Default Browser Proxy Settings 21-5
Configuring Custom Proxy Settings 21-5
Disabling the Use of a Proxy Server When Accessing the Internet 21-6
How to Set the Context Root for Web Services 21-6
How to Configure Connections to Use with Web Services 21-7
How to Work with Type Mappings 21-7
How to Choose Your Deployment Platform 21-8
How to Work with Web Services Code Insight 21-9
Working with Web Services in a UDDI Registry 21-10
How to Define UDDI Registry Connections 21-10
Creating UDDI Registry Connections 21-11
Editing the UDDI Registry Connections 21-11
Changing the View of UDDI Registry Connections 21-11
Refreshing the UDDI Registry Connections 21-12
Deleting the UDDI Registry Connections 21-12
What You May Need to Know About Choosing the View for your UDDI Registry Connection 21-12
Choosing the Business View 21-12
Choosing Category View 21-13
How to Search for Web Services in a UDDI Registry 21-13
How to Generate Proxies to Use Web Services Located in a UDDI Registry 21-13
How to Display Reports of Web Services Located in a UDDI Registry 21-14

How to Publish Web Services to a UDDI Registry 21-14
Creating JAX-WS Web Services and Clients 21-15
How to Create JAX-WS Web Services (Bottom-up) 21-15
Creating Java Web Services 21-16
Using Web Service Annotations 21-17
Using the Properties Window 21-17
Creating TopLink Database Web Service Providers 21-18
Regenerating Web Services from Source 21-18
Using Handlers 21-19
Handling Overloaded Methods 21-19
How to Create JAX-WS Web Services from WSDL (Top-down) 21-20
How to Create JAX-WS Web Service Clients 21-20
Creating the Client and Proxy Classes 21-21
Developing a JAX-WS Web Service Client Application 21-22
Referencing Web Services Using the @WebServiceRef Annotation 21-23
Enabling Web Service Atomic Transactions in a Web Service Client 21-24
Regenerating Web Service Client and Proxy Classes 21-24
Editing the Web Service Clients 21-25
Deleting the Web Service Clients 21-25
How to Use Web Service Atomic Transactions 21-25
Enabling Web Service Atomic Transactions in a Java Class 21-27
Enabling Web Service Atomic Transactions in the Properties Window 21-27
Enabling Web Service Atomic Transactions in a Web Service Client's Injectable Target 21-27
How to Use SOAP Over JMS Transport 21-28
Developing Web Services Using JMS Transport 21-28
Enabling JMS Transport in the Properties Window 21-29
Developing Web Service Clients Using JMS Transport 21-30
How to Use Fast Infoset for Optimizing XML Transmission 21-31
Configuring Fast Infoset on Web Services 21-31
Configuring Fast Infoset on Web Service Clients 21-33
Disabling Fast Infoset on Web Services and Clients 21-33
How to Use MTOM for Optimizing Binary Transmission 21-34
Enabling MTOM on Web Services 21-35
Enabling MTOM on Web Service Clients 21-36
Configuring MTOM on Web Services and Clients 21-36
How to Manage WSDL Files 21-36
Creating WSDL Documents 21-37
Displaying the WSDL for a Web Service 21-37
Adding a WSDL to a Web Service Project 21-37
Saving a WSDL to Your Local Directory 21-38
Viewing the WSDL Used to Create the Web Service Client 21-38

Refreshing the Local Copy of the WSDL and Regenerating the Web Service Client Proxy and Classes 21-39
Updating the Web Service WSDL Used by the Client at Run Time 21-39
How to Edit JAX-WS Web Services 21-42
How to Delete JAX-WS Web Services 21-43
Creating RESTful Web Services and Clients 21-43
How to Create RESTful Web Services 21-44
Example of a Simple RESTful Web Service 21-45
Creating a RESTful Web Service 21-45
Defining the Relative URI of the Root Resource and Subresources 21-46
Mapping Incoming HTTP Requests to Java Methods 21-49
Customizing Media Types for the Request and Response Messages 21-51
Extracting Information from the Request Message 21-52
Mapping HTTP Request and Response Entity Bodies Using Entity Providers 21-55
Accessing the RESTful Web Service WADL 21-55
How to Create RESTful Web Service Clients 21-57
Example of a Simple RESTful Client 21-58
Creating RESTful Web Service Clients 21-58
Creating WebSockets 21-59
How to Configure WebSockets in the Properties Window 21-59
How to Configure WebSockets Using Annotations 21-60
How to Test the WebSocket Connection 21-61
Attaching Policies 21-61
What You May Need to Know About OWSM Policies 21-61
What You May Need to Know About Oracle WebLogic Web Service Policies 21-62
How to Attach Policies to JAX-WS Web Service and Clients 21-62
Attaching Policies to JAX-WS Web Services 21-63
Attaching OWSM Policies to JAX-WS Web Service Clients 21-65
Overriding OWSM Policy Configuration Properties for the JAX-WS Web Service Clients 21-68
Invoking JAX-WS Web Services Secured Using WebLogic Web Service Policies 21-69
Editing or Removing Policies from JAX-WS Web Services 21-70
How to Attach Policies to RESTful Web Services and Clients 21-71
Attaching Policies to RESTful Web Services 21-71
Attaching Policies to RESTful Web Service Clients 21-72
Overriding OWSM Policy Configuration Properties for the RESTful Web Service Clients 21-74
Editing or Removing Policies from RESTful Web Services and Clients 21-74
How to Use a Different OWSM Policy Store 21-75
How to Use Custom Web Service Policies 21-75
Using Custom OWSM Policies 21-75
Using Custom Oracle WebLogic Web Service Policies 21-76
Deploying Web Services 21-76

How to Deploy Web Services to Integrated WebLogic Server 21-77
How to Deploy Web Services to a Standalone Application Server 21-77
How to Undeploy Web Services 21-78
Testing and Debugging Web Services 21-78
How to Test Web Services in a Browser 21-78
How to Debug Web Services 21-78
Debugging Web Services Locally 21-79
Debugging Web Services Remotely 21-80
Using the Data Window for Debugging Web Services 21-80
How to Test Web Services with JUnit 21-80
How to View Web Service Message Logs for an Application Server 21-81
Monitoring Web Services Network Traffic 21-81

22 Deploying Applications

About Deploying Applications 22-1
Developing Applications with the Integrated Application Server 22-3
Developing Applications to Deploy to Standalone Application Servers 22-4
Understanding the Archive Formats 22-4
Understanding Deployment Profiles 22-5
Understanding Deployment Descriptors 22-5
Configuring Deployment Using Deployment Plans 22-5
Deploying from the Java Edition 22-5
Running Java EE Applications in the Integrated Application Server 22-5
Understanding the Integrated Application Server Log Window 22-7
Rules Governing Deployment to the Integrated Application Server 22-7
Working with Integrated Application Servers 22-8
How to Create a New Integrated Application Server Connection 22-8
How to Run and Debug with an Integrated Application Server 22-9
Working with the Default Domain 22-9
One-Click Running of Applications in the Integrated Application Server 22-10
How to Start the Integrated Application Server 22-11
How to Cancel a Running Deployment 22-12
How to Terminate an Integrated Application Server 22-12
How to Configure Startup and Shutdown Behavior for Integrated Application Servers 22-12
How to Log In to the Integrated WebLogic Server Administration Console 22-13
Connecting and Deploying Java EE Applications to Application Servers 22-13
How to Create a Connection to the Target Application Server 22-14
Connecting to Specific Application Server Types 22-16
Connecting to Oracle WebLogic Server 22-16
Connecting to GlassFish 22-16
Connecting to JBoss 22-16

Connecting to Tomcat 22-17
Connecting to WebSphere Server 22-17
How to Create and Edit Deployment Profiles 22-18
About Deployment Profiles 22-18
Creating Deployment Profiles 22-20
Viewing and Changing Deployment Profile Properties 22-21
Configuring Deployment Profiles 22-21
How to Create and Edit Deployment Dependencies 22-22
About Library Dependencies 22-22
How to Create and Edit Deployment Descriptors 22-24
Creating Deployment Descriptors 22-26
How to Create a Web Service Policy Reference 22-27
Viewing or Modifying Deployment Descriptor Properties 22-27
How to Configure Global Deployment Preferences 22-28
How to Configure Applications for Deployment 22-28
How to Configure an Application for Deployment to Oracle WebLogic Server 22-28
How to Configure a Client Application for Deployment 22-29
How to Configure an Applet for Deployment 22-29
Setting Up JDBC Data Sources on Oracle WebLogic Server 22-30
Preparing an Application for Deployment to a Third Party Server 22-31
How to Use Deployment Plans 22-32
How to Create and Use Deployment Plans 22-33
How to Generate Deployment Plans 22-34
Deploying Java Applications 22-35
Deploying to a Java JAR 22-35
Deploying to an OSGi Bundle 22-36
Deploying Java EE Applications 22-36
How to Deploy to the Application Server from JDeveloper 22-36
How to Deploy a RAR File 22-37
How to Add a Resource Adapter Archive (RAR) to the EAR 22-37
How to Deploy a Metadata Archive (MAR) File 22-38
How to Deploy an Applet as a WAR File 22-38
How to Deploy a Shared Library Archive 22-39
How to Deploy to a Managed Server That Is Down 22-39
Post-Deployment Configuration 22-40
Testing the Application and Verifying Deployment 22-40
Deploying from the Command Line 22-41
ojdeploy 22-41
Using ojdeploy from Mac OS X Platforms 22-44
Using ojdeploy 22-44
How to Override Without Editing a Build Script 22-45
How to Deploy Multiple Profiles from the Command Line 22-45

How to Use Wildcard Samples 22-46
How to Create a Log File for Batch Deployment 22-47
Timeouts 22-48
How to Deploy from the Command Line Using Ant 22-48
How to Generate an Ant Build Script 22-49
About The build.xml File 22-50
About The build.properties File 22-50
ojdeploy for Extension Developers 22-52
Using ojserver 22-55
Deploying Using Java Web Start 22-55
Purpose of the Java Web Start Technology 22-56
Files Generated by the Create Java Web Start-Enabled Wizard 22-57
Role of the Web Server in JDeveloper 22-57
How to Create a Java Web Start File 22-58
How to Create a Java Client Web Archive for Java Web Start 22-58
How to Create a Java Web Start JNLP Definition for Java Clients 22-59
How to Deploy a Java Client Web Application Archive for Java Web Start 22-60
Deploying Using Weblogic SCA Spring 22-61
About WebLogic SCA 22-61
About Spring 22-62
Installing the Weblogic SCA Spring Extension 22-62
Using Oracle WebLogic SCA 22-62
How to Create WebLogic SCA Projects 22-62
How to Edit Oracle WebLogic SCA Definition Files 22-63
How to Deploy WebLogic SCA Applications to Integrated WebLogic Server 22-65
How to Deploy WebLogic SCA Applications to Oracle WebLogic Server 22-65
Using Spring 22-66
How to Create Spring Bean Applications 22-66
What Happens When You Create a Spring Bean Configuration File 22-66
Troubleshooting Deployment 22-66
Common Deployment Issues 22-67
[Deployer: 149164] The domain edit lock is owned by another session in exclusive mode - hence this deployment operation cannot proceed 22-67
How to Troubleshoot Deployment to Integrated Application Servers 22-67
Stopping Integrated Application Server 22-67
Running Out of Memory 22-67
Reinstalling JDeveloper in a Different Location 22-68
How to Troubleshoot Deployment to Oracle WebLogic Server 22-68
ORA-01005: null password given; logon denied 22-68
ORA-01017: invalid username/password; logon denied 22-68
[Oracle JDBC Driver] Kerberos Authentication was requested, but is not supported by this Oracle Server 22-68

Application Does Not Work After Creating a Global Data Source from the Oracle WebLogic Server Administration Console 22-68
Redeploying an Application to a Server that is Down 22-68
Attempting to Deploy to a Server that No Longer Exists 22-69
Deploying to a remove server fails with HTTP Error Code 502 22-69
No Credential Mapper Entry Found 22-69
How to Troubleshoot Deployment to IBM WebSphere 22-70
Deployment Fails When EAR Contains Spaces 22-70
Deployment Fails When the Path to the WebSphere Server Contains Spaces 22-70
Application Displays Administrative Console User Name 22-70

23 Getting Started with Working with Databases

About Working with Databases 23-1
Connecting to and Working with Databases 23-1
Designing Databases 23-1
Getting Started With Oracle Database Express Edition 23-2
How to Manage Database Preferences and Properties 23-3

24 Using the Database Tools

Using the Databases Window 24-1
Using the Database Cart 24-3
Using the Structure Window 24-5
Using the Database Reports Window 24-7
Using the Find Database Object Window 24-8
Using the SQL Worksheet 24-9
Using Execution Plan 24-12
How to Recall Statements from the SQL Worksheet History 24-12
Using the SQL History Window 24-13
Using the Snippets Window 24-13
Using the Database Object Viewer 24-14
Database Object Viewer Tabs Toolbars 24-14
Using the PL/SQL Source Editor 24-16
Using Test Query 24-17
Synchronizing Package Specifications and Bodies 24-19
Using SQL*Plus 24-20
DBMS Output Window 24-21
OWA Output Window 24-22

25 Connecting to and Working with Databases

About Connecting to and with Working with Databases 25-1
Configuring Database Connections 25-1
- Connection Scope 25-2
- What Happens When You Create a Database Connection 25-2
- About Connection Properties Deployment 25-3
- How to Create Database Connections 25-3
  - Create a CloudDB/ATPDB Database Connection 25-3
- Defining Additional JDBC Parameters 25-6
- Using Different Drivers 25-7
- Connecting to Oracle Database Using OCI8 25-8
- How to Edit Database Connections 25-8
- How to Export and import Database Connections 25-9
  - Exporting Database Connections 25-9
  - Importing Database Connections 25-9
- How to Open and Close Database Connections 25-10
- How to Delete Database Connections 25-10
- How to Register a New Third-Party JDBC Driver 25-10
- How to Create User Libraries for Non-Oracle Databases 25-11
- Reference: Connection Requirements for Oracle's Type 2 JDBC Drivers (OCI) 25-11

Browsing and Searching Databases 25-12
- Browsing Databases 25-12
  - Browsing Online Databases 25-13
  - Browsing Offline Database Objects 25-13
  - How to View Online and Offline Database Objects 25-13
- How to Browse online Database Objects 25-13
- How to Browse Offline Databases and Schemas 25-14
- How to Use Database Filters 25-14
- How to Enable and Disable Database Filters 25-15
- How to Open a Database Table in the Database Object Viewer 25-15
- How to Edit Table Data 25-16
- How to Find Objects in the Database 25-16

Connecting to Databases 25-16
- What Happens When You Create a Connection to a Database 25-17
- How to Create Connections to Oracle Databases 25-17
  - How to Create a Connection to Oracle Database 25-17
  - How to Create a Connection to MySQL 25-17
  - How to Create a Connection to Oracle TimesTen In-Memory Database 25-18
- How to Create Connections to Non-Oracle Databases 25-19
  - How to Create a Connection to Java DB/Apache Derby 25-19
  - How to Create a Connection to IBM DB2 Universal Database 25-21

How to Create a Connection to IBM Informix Dynamic Server 25-22
How to Create a Connection to Microsoft SQL Server 25-22
How to Create a Connection to SQLite 25-24
How to Create a Connection to Sybase ASE 25-25
Connecting and Deploying to Oracle Database Cloud Service 25-26
Types of JDeveloper Connection to Oracle Database Cloud Service 25-26
Creating an Oracle Database Cloud Service Connection 25-26
Editing an Oracle Database Cloud Service Connection 25-27
Connecting and Disconnecting from an Oracle Database Cloud Service Connection 25-27
Using the Database Cart 25-28
Setting the Default Directories Used By the Database Cart 25-28
Configuring Database Cart Tools 25-28
Deploying to Oracle Database Cloud Service 25-29
Deploying to a Database Deployment File 25-30
Saving a Database Cart 25-30
Opening a Saved Database Cart 25-31
Examining Deployments in an Oracle Database Cloud Service Connection 25-31
Importing and Exporting Data 25-32
Importing Data Using SQL*Loader 25-32
Importing Data Into an External Table 25-32
How to Import Data into Existing Tables 25-33
How to Import Data to New Tables 25-33
How to Import Data Using SQL*Loader 25-33
How to Import Data Using External Tables 25-34
Exporting Data from Databases 25-34
How to Export Data to Files 25-35
Copying, Comparing, and Exporting Databases 25-35
How to Copy Databases 25-35
How to Compare Database Schemas 25-36
How to Export Databases 25-36
Working with Database Reports 25-36
Using Database Reports 25-36
How to Run Database Reports 25-37
How to View the SQL for a Report 25-37
How to Create User-Defined Database Reports 25-37
How to Edit User-Defined Database Reports 25-38
How to Create Reports Folders 25-38
How to Export User-Defined Reports 25-38
How to Import User-Defined Reports 25-38
Reference: Pre-Defined Database Reports 25-39
Troubleshooting Database Connections 25-44

Deploying to a Database that Uses an Incompatible JDK Version 25-44

26 Designing Databases Within Oracle JDeveloper

About Designing Databases Within Oracle JDeveloper 26-1
Creating, Editing, and Dropping Database Objects 26-1
Working with Offline Database Definitions 26-1
Offline Databases 26-3
Configuring Offline Database Emulation 26-4
How to Create Offline Databases 26-4
Offline Schemas 26-5
How to Create Offline Schemas 26-5
How to Create Offline Database Objects 26-5
How Reverse Engineer Database Definitions Based on Database Objects 26-10
Offline Tables and Foreign Keys 26-11
How to Refresh Offline Database Objects 26-12
How to Create Objects from Templates 26-12
Working with User Property Libraries 26-13
How to Generate Offline Database Objects to the Database 26-14
Renaming Offline Database Objects 26-16
Using Offline Database Reports 26-17
Transforming from a UML Model 26-19
Working with Offline Database Objects in Source Control Systems 26-19
Working with Database Objects 26-20
Using Database Reports 26-21
Validating Date and Time Values 26-21
Creating Scripts from Offline and Database Objects 26-21
How to Create SQL Scripts 26-21
How to Create OMB Scripts from Tables 26-22
How to Create SXML Scripts 26-23

27 Using Java in the Database

About Using Java in the Database 27-1
Choosing SQLJ or JDBC 27-1
Using SQLJ 27-1
Using Oracle JDBC Drivers 27-2
SQLJ versus JDBC 27-3
Embedding SQL in Java Programs with SQLJ 27-4
How to Create SQL Files 27-4
How to Create SQLJ Classes 27-4
How to Compile SQLJ Classes 27-4

How to Use Named SQLJ Connection Contexts 27-5
How to Declare a SQLJ Connection Context Class 27-5
How to Create a Connection Context Object 27-5
How to Debug SQLJ Classes 27-5
How to Set SQLJ Translator Options 27-5
How to Use SQLJ Connection Options 27-6
Embedding SQL in Java Programs with JDBC 27-7
How to Choose a JDBC Driver 27-7
How to Modify a Project to Use a Non-Default JDBC Driver 27-8
How to Code a JDBC Connection 27-8
Using Java Stored Procedures 27-9
How to Debug Java Stored Procedures 27-16
How to Remove Java Stored Procedures 27-17

28 Running and Debugging PL/SQL and Java Stored Procedures

About Running and Debugging PL/SQL and Java Stored Procedures 28-1
Running and Debugging Functions, Procedures, and Packages 28-1
Debugging PL/SQL Programs and Java Stored Procedures 28-2
Debugging PL/SQL Objects 28-2
PL/SQL objects you can debug with JDeveloper 28-3
What You May Need to Know 28-3
Appearance of debug information in supported Oracle Database 28-4
How to Specify the Database Debugger Port 28-4
Debugging PL/SQL and Java Stored Procedures Prerequisites 28-4
Prerequisites for Debugging PL/SQL and Java Stored Procedures 28-5
Prerequisites for Debugging Java Stored Procedures 28-5
How to Locally Debug PL/SQL Programs 28-5
How to Remotely Debug PL/SQL Programs 28-6
Using Acceptable Legal PL/SQL Expressions in the Debugger 28-8

Preface

Documentation for Oracle JDeveloper users that describes how to use the JDeveloper IDE and provides detailed information on the functionality available within it.

Audience

This document is intended for developers that use Oracle JDeveloper and 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 customer access to and use of Oracle support services will be pursuant to the terms and conditions specified in their Oracle order for the applicable services.

Diversity and Inclusion

Oracle is fully committed to diversity and inclusion. Oracle respects and values having a diverse workforce that increases thought leadership and innovation. As part of our initiative to build a more inclusive culture that positively impacts our employees, customers, and partners, we are working to remove insensitive terms from our products and documentation. We are also mindful of the necessity to maintain compatibility with our customers' existing technologies and the need to ensure continuity of service as Oracle's offerings and industry standards evolve. Because of these technical constraints, our effort to remove insensitive terms is ongoing and will take time and external cooperation.

Related Documents

- *Installing Oracle JDeveloper*
- *Understanding Oracle Application Development Framework*
- *Developing Extensions for Oracle JDeveloper*
- *Developing Fusion Web Applications with Oracle Application Development Framework*
- *Developing Web User Interfaces with Oracle ADF Faces*
- *Developing Applications with Oracle ADF Data Controls*
- *Oracle JDeveloper 14c Online Help*
- *Oracle JDeveloper 14c Release Notes*, link included with your Oracle JDeveloper 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

The following topics introduce the new and changed features of Oracle JDeveloper and other significant changes that are described in this guide, and provides pointers to additional information. This document is the new edition of *Developing Applications with Oracle JDeveloper*.

New and Changed Features for 14*c* (14.1.2.0.0)

Oracle JDeveloper 14*c* (14.1.2.0.0) includes the following new and changed features for this document.

- Ability to create a CloudDB/ATPDB connection to an Oracle Cloud Database using Oracle Cloud Wallet. See Create a CloudDB/ATPDB Database Connection.

1

Introduction to Oracle JDeveloper

JDeveloper is an integrated development environment (IDE) for building applications. It builds applications using the latest standards for Java, XML, Web services, and SQL. This chapter provides an overview of Oracle JDeveloper. It includes the following sections.

- About Oracle JDeveloper
- Oracle JDeveloper Information Resources
- Configuring Proxy Settings
- Migrating to Oracle JDeveloper 14*c*

About Oracle JDeveloper

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

JDeveloper supports the complete development life cycle with integrated features for modeling, coding, debugging, testing, profiling, tuning, and deploying applications. It provides a visual and declarative development approach and works together with the Oracle ADF to simplify development. It is the main development platform for the Oracle Fusion Middleware suite of products.

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.

Key features 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.

Oracle JDeveloper Information Resources

Oracle JDeveloper resources such as documentation, online demonstrations, tutorials, sample applications, and the Oracle JDeveloper Forum help you get started with application development.

Oracle JDeveloper includes resources designed to get you up and running quickly. 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. The following table lists several of these resources.

Table 1-1 Supporting Oracle JDeveloper Resources

Resource	Description
Oracle JDeveloper	The main Oracle page for Oracle JDeveloper is located at: https://www.oracle.com/application-development/technologies/jdeveloper.html
Oracle JDeveloper documentation	JDeveloper documentation is located at: https://docs.oracle.com/en/middleware/developer-tools/jdeveloper/index.html
Oracle JDeveloper tutorials	The tutorials provide step-by-step instructions to accomplish specific tasks in Oracle JDeveloper. The tutorials are located at: http://docs.oracle.com/cd/E37547_01/tutorials/toc.htm
Sample applications	The Summit sample applications for Oracle ADF are a set of applications developed with the purpose of demonstrating common use cases in ADF applications, including the integration between the components of the Oracle ADF technology stack (ADF Business Components, ADF Faces, ADF DVT Faces, and ADF Controller). The samples consist of several workspaces that demonstrate various features of component functionality. For descriptions of the sample code drawn from the Summit sample applications, see Introduction to the ADF Sample Application and other relevant sections in *Developing Fusion Web Applications with Oracle Application Development Framework*. The sample applications are available from: https://www.oracle.com/tools/downloads/adf1221samples-downloads.html
Oracle forums	You can use Oracle Forums to ask a question, contribute to a discussion, or interact with other users. For JDeveloper information, use this URL: https://forums.oracle.com/ords/apexds/domain/dev-community/category/java_development_tools?tags=application-development-in-java,jdeveloper-and-adf

Configuring Proxy Settings

Use the procedure to configure proxy settings if you need to customize the system proxy settings to external servers.

By default JDeveloper uses the system proxy settings for your device. You might need to customize these settings to reach external servers.

To configure proxy settings:

1. Choose **Tools > Preferences > Web Browser and Proxy**.
2. Select the **Proxy Settings** tab.
3. Select a proxy option and fill in any active fields.

   No Proxy. Select this option when your system does not use a proxy to access the internet.

Use System Default Proxy Settings. Select this option to have the server use the default proxy settings on your machine. These are the settings configured in your OS (for Windows or Mac) or in your Window Manager. This includes the host, the port, and the exceptions, and you cannot add to or enhance those settings.

The expectation is that your OS settings are correct and there is nothing else to add. To edit the settings for your operating system:

- On Windows, go to Control Panel, Internet Options.
- On MacOS go to System Preferences, Network, Advanced, Proxies.
- On Linux, the proxy settings are configured in the window manager (for example, on Gnome, this is in System, Preferences, Network Proxy).

Use Automatic Configuration Script. Specify the location of an automatic configuration script. For example, the URL for a corporate `wpad.dat` file (`http://wpad.myhost.com/wpad.dat`).

Manual Proxy Settings. Manually define the proxy settings specifically for your organization. Like the System Defaults settings, the manual No Proxy settings are initially inherited from the OS. Once inherited, they can be modified, enhanced, or even replaced.

4. Click **Test Proxy** to verify that any settings you provided are correct. If the test fails, verify that you have entered the correct URL, host information, or authentication information. and that you can access the external server through your network or VPN.
5. When you have verified that you can connect to your proxy server, click **OK**.

Using an Automatic Configuration Script for Proxy Settings

Use the procedure to configure JDeveloper to use an automatic configuration script for proxy settings.

If your organization uses an automatic configuration script for proxy settings (for example, `wpad.dat` or similarly named scripts), you can configure JDeveloper to use this script automatically.

To configure JDeveloper to use an automatic configuration script:

1. Choose **Tools > Preferences > Web Browser and Proxy**.
2. Select the **Proxy Settings** tab.
3. Select the option Use Automatic Configuration Script.
4. In the **Script** field, enter the complete URL to the server on which the script resides.
5. Click **Test Proxy** to verify that the URL is correct. If the test fails, verify that you have entered the correct URL, and that you can access that URL through your network or VPN.
6. When you have verified that you can connect to your proxy server, click **OK**.

JDeveloper will now automatically use the script at the specified URL for proxy settings.

Migrating to Oracle JDeveloper 14*c*

You can migrate applications and projects, or preferences and settings, from earlier versions of JDeveloper.

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 14*c*, see Migrating Oracle JDeveloper From a Previous Version in *Installing Oracle JDeveloper*.

2

Oracle JDeveloper Accessibility Information

Oracle JDeveloper provides a wide range of features that are designed to support accessibility. Our goal is to make Oracle Products, Services, and supporting documentation accessible to the disabled community.
This chapter includes the following sections:

- Using a Screen Reader and Java Access Bridge with Oracle JDeveloper
- Oracle JDeveloper Features that Support Accessibility
- Recommendations for Customizing Oracle JDeveloper
- Highly Visual Features of Oracle JDeveloper

Using a Screen Reader and Java Access Bridge with Oracle JDeveloper

Install Java Access Bridge on Windows computers to work with assisting technologies.

For assisting technologies, like screen readers, to work with Java-based applications and applets, the Windows-based computer must also have Java Access Bridge installed. Refer to Using a Screen Reader and Java Access Bridge with Oracle JDeveloper in *Installing Oracle JDeveloper* for the screen reader setup procedure, and for the recommended minimum technology stack.

Oracle JDeveloper Features that Support Accessibility

The Oracle Accessibility Program web page provides information on how to configure and use the accessibility features of Oracle JDeveloper.

Oracle JDeveloper supports accessibility features. For additional accessibility information about Oracle products, including information on how to configure and use them, see the Oracle Accessibility Program page at:

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

Oracle's goal is to ensure that disabled end-users of our products can perform the same tasks, and access the same functionality as other users. Oracle JDeveloper provides a number of features that are designed to support accessibility goals.

Keyboard Access

Oracle JDeveloper features support keyboard access to several menu and toolbar functionality so that users may navigate between windows and work with them, or invoke resources.

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 windows 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 Applications window, Structure window, source editor, Properties window, 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 Properties window.
- 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.

Screen Reader Readability

Screen readers in JDeveloper ensure that menus and menu items, toolbar items, hint text, open windows and components, status text, controls within wizards, dialogs, and runtime applications are read.

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 window 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.

Flexibility in Font and Color Choices

For users who are visually challenged, Oracle JDeveloper offers options in font color, font size, and background color on user interfaces.

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.

No Audio-only Feedback

JDeveloper ensures that when you work in the IDE audio feedback is supplemented with visual indicators.

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.

No Dependency on Blinking Cursor and Animation

By limiting its dependence on the use of a blinking cursor and animation, JDeveloper ensures greater feature accessibility.

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.

Screen Magnifier Usability

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

How to Change the Editor or Tabbed View of a File

In JDeveloper, use accelerators to change the Editor view or Tabbed view of a file.

When you press Enter on a node in the Applications window, 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.

How to Read Text in a Multi-line Edit Field

By selecting text and moving the mouse cursor up or down, you can get a screen-reader to read the 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.

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.

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

Save the contents of the Log window as an HTML file, and add it to a project or application as a resource so that JDeveloper displays links correctly.

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 as a resource. Open the file from the Applications window 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.

Recommendations for Customizing Oracle JDeveloper

JDeveloper offers features to customize keyboard usage, display attributes of the IDE, and timing.

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.

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.

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.

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.

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.

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.

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.

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.

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.

Highly Visual Features of Oracle JDeveloper

JDeveloper supports some very visual features. Visually challenged users can access the UI, the visual editors, and the Components window using the equivalent functionality in 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 Components window. The source editor provides equivalent functionality, as elements and tags that can be selected from the Components window can also be entered in the source editor.

You can add a component from the Components window 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 help you 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:

- About Working with Oracle JDeveloper
- Working with JDeveloper Roles
- How to Manage JDeveloper Features and Installed Updates
- Working with Windows in the IDE
- Keyboard Navigation in JDeveloper
- Customizing the IDE
- Working with the Resources Window
- Working with Source Files
- Working with Extensions
- Using the Online Help
- Common Development Tools
- Adding External Tools to JDeveloper
- Working with Tasks
- Working with the Tasks Window

About Working with Oracle JDeveloper

Oracle JDeveloper is a cross-platform IDE that is available in two editions—Oracle JDeveloper Studio with all features and Oracle JDeveloper Java with core Java and XML features.

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

Working with JDeveloper Roles

The roles that JDeveloper provides tailor the environment to development needs. The selected role decides which features and options are available for development.

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.

How to Change the JDeveloper Role

The role that you selected at JDeveloper start can be changed. Use the procedure to change the selected role with the **Switch Roles** option from the Tools menu.

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.

How to Manage JDeveloper Features and Installed Updates

JDeveloper provides the functionality to disable features or uninstall installed updates so that you see only those components of the IDE that are relevant to the development. Use procedure to add or remove features.

To optimize performance and user experience, JDeveloper allows you to disable features you do not 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. Additionally, you can also uninstall updates that you previously installed.

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 and Updates 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 features. Click a feature or feature category to view its description on the right.

3. Check the features you want to add or keep, 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 Cache**. When you clear the cache, the features are not loaded automatically each time you restart JDeveloper. The features are loaded only when you use them.
5. Click **Update Status** when you are done. This feature allows you to commit changes to the features you have made.

> **Note:**
>
> **Update Status** is not applicable when you install features using **Check For Updates** feature.

To uninstall an installed update:

1. From the main menu, select **Tools > Features**. The Manage Features and Updates dialog opens.
2. Click **Installed Updates** to view the updates that have been installed.
3. Search for the update you want to uninstall by entering it in the **Search** field, or scroll in the list of updates. Click an update or update category to view its description on the right.
4. Check the updates you want to uninstall and click **Uninstall**.

Working with Windows in the IDE

You can organize Dockable windows and Tabbed editor windows in JDeveloper.

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.

How to Maximize Windows

Double-clicking the title bar of a JDeveloper window maximizes the screen to full screen view, or reverts to the earlier view.

You can maximize any JDeveloper window for better visibility and convenience. 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.

Windows do not stay maximized when JDeveloper is closed and then reopened. Instead the window returns to its default size. However minimized windows stay minimized when JDeveloper is reopened.

How to Minimize and Restore Dockable Windows in the IDE

Use the Minimize button to minimize or maximize a dockable window in JDeveloper, or set it to remain open in its place.

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.

How to Dock Windows in the IDE

Windows can be docked singly or in groups, floated, resized, and positioned anywhere within JDeveloper, or tabbed together in one location. Use the tabulated actions to move dockable windows, to decouple one tabbed window from a group, or to reposition docked windows.

You can float any window that's normally docked—the Applications window, any custom window, the Log window, the Properties window, the Components window. 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 windows on a smaller screen they can sometimes be hidden by other information windows as you work.

All of the tools available under the Window menu—the Applications window, Structure window, Properties window, 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

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

How to Close and Reopen Dockable Windows in the IDE

You can easily open and close the main elements of the JDeveloper IDE. These include the Applications window, Databases window, Structure window, Properties window, Components window, Resources window and Log window.

Opening a Closed Window

Select a window from the Window menu to open the window.

You can open a window that is currently closed.

To open a closed window:

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

Closing an Open Window

Use **Close** in any open window to close the window. Alternatively, use keyboard shortcuts.

You can close a window that is currently open.

To close an open window, perform one of the following steps:

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

How to Restore Window Layout to Factory Settings

The **Restore Windows to Factory Settings** option in the Window menu restores the layout of a window.

You can restore the window layout in JDeveloper to the default, factory setting.

To restore the layout of dockable windows:

- From the **Window** menu, select **Restore Windows to Factory Settings**.

Keyboard Navigation in JDeveloper

Keyboard navigation can perform any task that can be accomplished with a mouse device.

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 that you can using the mouse.

How to Work with Shortcut Keys in the IDE

JDeveloper provides several predefined and customizable keyboard schemes that are overlaid over the shortcut keys defined in the Java Look and Feel guidelines. The JDeveloper scheme prevails when a shortcut key is available in both the look and feel guidelines and the JDeveloper keyboard scheme.

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.

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.

Loading Preset Keyboard Schemas

The interaction of the currently enabled scheme with the Java look and feel guidelines determine the shortcut keys enabled in JDeveloper. Use the procedure to load preset keyboard schemes.

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 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**.

Viewing JDeveloper Commands and Associated Keyboard Shortcuts

Use the procedure to navigate from the Tools menu and view JDeveloper commands and their associated keyboard shortcuts in the **Available Commands** list.

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.

Redefining a Keyboard Shortcut for a Command

Use the procedure to navigate from the Tools menu to the **New Shortcut** field, and change the shortcut currently assigned for a command in a keyboard scheme.

If you prefer using a different keyboard shortcut to the one currently assigned for a command in a keyboard scheme, you can specify a shortcut of your choice.

> **Note:**
>
> If you use the Smart Common Input Method (SCIM), the keyboard shortcut for Completion Insight (**Ctrl+Space**) will not function as expected in the JDeveloper Code Editor, because **Ctrl+Space** is interpreted as part of the SCIM user interface. If you need to use SCIM, you should follow the instructions shown here to map a different key sequence to Completion Insight.

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**.

Importing and Exporting Keyboard Schemas

Use the procedure to navigate from the Tools menu to the **Export** or **Import** options and import or export keyboard schemas.

JDeveloper enables you to import and export keyboard schemas.

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.

Common Navigation Keys

Use the table to understand how you can navigate in JDeveloper using 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
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 panel which may be an editor, or a window, 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

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

Key	Cursor Movement	Ctrl+cursor Movement
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

Navigation in Standard Components

Use the table to acquaint yourself with the keyboard navigation for UI components in JDeveloper such as buttons, checkboxes, dropdown lists, and combo boxes.

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
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)

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

Navigation	Keys
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
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

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

Navigation	Keys
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 between radio button	Arrow 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	Shift+Up Arrow Key
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

Table 3-8 (Cont.) Keyboard Navigation for Sliders

Navigation	Keys
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
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

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

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

Navigating Complex Controls

Use keyboard shortcuts for complex UI components including menu bar, panels, tables, tabs in dockable windows, views and dialogs, trees, and wizards.

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 panel contents if focus moves beyond visible area of canvas.

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

Navigation	Keys
Move to start/end of data (component buttons)	Ctrl+Home, Ctrl+End
Select an element	Enter or Spacebar
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

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

Navigation	Keys
Navigate to panel header from contents (when focus is on top item in list)	Up Arrow
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

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

Navigation	Keys
Extend selection to beginning of column	Ctrl+Shift+Home
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 navigational 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

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

Navigation	Keys
Expand (if item contains children)	Right Arrow
Collapse (if item contains children)	Left Arrow
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

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

Navigation	Keys
Navigate backward between components on wizard panel, wizard navigation bar buttons, and navigation panel	Shift+Tab
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

Navigation in Specific Components

Use keyboard shortcuts for JDeveloper-specific UI components such as dialogs, overview editors, windows such as the Components and Properties windows, panel elements of text editors, text or canvas areas of text editors, graphical editors and canvas areas of graphical editors, and the action to close the Monitors and the Inspector windows while debugging.

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

Dialogs

The following table describes the keyboard actions to perform navigational 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

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

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

Component and Resources windows

The following table describes the keyboard actions to perform navigational tasks in windows such as the Components and Resources windows.

Table 3-20 Keyboard Navigation for Components and Resources windows

Navigation	Keys
Navigate forward in or out of window	Ctrl+Tab This moves you into first item within the pane.
Navigate backward in or out of window	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

Windows such as the Applications Window, Databases Window, Applications Server Window

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

Table 3-21 Keyboard Navigation for Windows such as the Applications Window, Databases Window, and Application Server Window

Navigation	Keys
Navigate forward in or out of window	Ctrl+Tab This moves you into first item within the pane.
Navigate backward in or out of window	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
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

Properties window

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

Table 3-22 Keyboard Navigation for the Properties window

Navigation	Keys
Navigate forward into or out of Properties window	Ctrl+Tab
Navigate backward into or out of Properties window	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

Table 3-22 (Cont.) Keyboard Navigation for the Properties window

Navigation	Keys
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 panel 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

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

Navigation	Keys
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
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

Table 3-26 (Cont.) Keyboard Navigation for Canvas Areas of Graphical Editors

Navigation	Keys
Activate context menu	Shift+F10

Monitors and Inspector Windows

The following table describes the keyboard action to close the Monitors and the Inspector windows while debugging.

Table 3-27 Keyboard Navigation for the Monitors and Inspector Windows

Navigation	Keys
Closes the Monitors and the Inspectors windows without using the standard keyboard operations. The shortcut opens the context menu for the child window which provides the **Close** button.	Ctrl + Spacebar

Customizing the IDE

In JDeveloper, you can customize the general environment, dockable windows, and the Components and Compare windows.

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

- Change the look and feel
- Customize the general environment
- Customize dockable windows
- You can also customize the following windows in the IDE:
  - Compare window
  - Components window

How to Change the Look and Feel of the IDE

Use the procedure to use the Look and Feel option for Environment to change the appearance of JDeveloper using pre-defined settings.

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.

How to Customize the General Environment for the IDE

Use the procedure to change the general environment settings for the IDE. You can also define 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.

You can customize the default display options, In addition, you can define 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.

How to Customize the Compare Window in the IDE

Use the procedure to customize the display of the Compare window from **Preferences**.

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**.
3. On the Compare page, set the options available for the display of two files being compared.
4. Click **OK**.

How to Customize the Components window

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

How to Add a Page to the Components Window

Use the procedure to add a page to the Components Window.

You can add pages to the Components window, within which to group additional components. You can also add components to existing pages.

To add a page to the Components Window:

1. From the main menu, choose **Tools** > **Configure Components** to open the Configure Components dialog. For more information at any time, press F1 or click **Help** from within the Configure Components dialog.
2. Optionally, in the Configure Components dialog, for **Page Type** select the appropriate type to limit the display in the **Pages** list.
3. In the Configure Components dialog, underneath the **Pages** list box, click **Add**.
4. In the Create Components Window 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 Components Window dialog.
6. If finished, click **OK**. The new page is now added to the dropdown list in the Components window. It also appears in the Pages list of the Configure Components window dialog.

How to Add a JavaBeans Component to the Components Window

Use the procedure to add a JavaBeans component to the Components Window so that it can be selected from the Components window and inserted into any open file in the Java Visual Editor.

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

To add a JavaBeans component to the Components Window:

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 Components** to open the Configure Components dialog. For more information at any time, press F1 or click **Help** from within the Configure Components dialog.
3. Optionally, in the Configure Components 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 Components 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 Components 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 Components window dialog.

10. If finished, click **OK**.

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

How to Remove a Page from the Components Window

Use the procedure to remove a page from the Components Window so that it no longer appears in the list of the Components window or the Pages list of the Configure Components window .

You can remove pages from the Components window.

To remove a page from the Components Window:

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

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

How to Remove a Component from the Components Window

Use the procedure to remove a component from the list in Components or the Components list in the Configure Components Window dialog.

You can remove a component from the Components window.

To remove a component from the Components Window:

1. From the main menu, choose **Tools** > **Configure Components** to open the Configure Components Window dialog. For more information at any time, press F1 or click **Help** from within the Configure Components Window dialog.
2. Optionally, in the Configure Components Window 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 Components Window dialog, click **OK**.

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

You cannot remove a component using the Components Window context menu. You must work through the Configure Components Window dialog.

How to Change Roles in JDeveloper

Use the step to change the role for JDeveloper that becomes available upon a restart. Changing roles changes the environment in JDeveloper to suit development needs.

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.

How to Associate File Types with JDeveloper

Use the procedure to associate a file type with JDeveloper so that opening a file of that type on the Windows platform automatically launches 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**.

Working with the Resources Window

Use the Resources window to create connections to resources, such as application servers, databases, and WebDAV servers so that they can be used in different applications and shared with other users.

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

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 be 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.

Using the Resources Window

Use the Resources window to locate resources stored in repositories through IDE connections. Locate resources by browsing, or search for resources and save searches.

By default, the Resources window is displayed to the right of the JDeveloper window. The Resources window 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

How to Open the Resources Window

Use the **Resources** option for Window to open the Resources window, and work with resources such as application servers, databases, and WebDAV servers.

The Resources window allows you to work with different resources, such as application servers, databases, and WebDAV servers.

To open the Resources window:

- In the main menu, choose **Window > Resources**.

How to Refresh the Resources Window

Follow the step to refresh the Resources window using either the **Refresh** option in Resources, or the context menu for that window.

The Resources window allows you to work with different resources, such as application servers, databases, and WebDAV servers.

To refresh the Resources window:

- In the Resources window, click **New** and choose **Refresh**.

  Alternatively, in the Resources Window choose **Refresh** from the context menu of an object in the My Catalogs panel or the IDE Connections panel.

Working with IDE Connections

Connections in JDeveloper can be IDE connections that can be reused in different applications, or shared between users, or application connections that are available only to one

application. Use the procedures to create an IDE connection, edit an IDE connection. or to add a connection to an application.

When you create a connection in JDeveloper, you can create it as an IDE connection that can be reused in different applications, or shared between users, or as an application connection where the connection is only available to that application.

IDE connections are globally defined connections available for reuse, and they are listed in the IDE Connections panel of the Resources window. You can copy IDE connections to the Applications window to use them within an application.

IDE connections are listed in the IDE Connections panel of the Resources window. In addition, some types of connections may appear in special connection-type windows. For example, database connections are also listed in the Databases window under the IDE Connection node, and you edit database objects through the database connection in the Databases window.

The different types of connections that can be made depends on the technologies and extensions available to you. To see what you can create a connection to, choose **IDE Connections** from the **New button** in the Resources window. The specific types of connection you can make depend on the technologies and extensions available to you.

The file system location for Resources connection descriptor definition information is

system-dir/jdeveloper/system12.1.2.*n.nn.nn.nn*/o.jdeveloper.rescat2.model/connections/connections.xml

To create an IDE connection:

1. In the IDE Connections panel of the Resources window, choose **IDE Connections** 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.

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

To edit an IDE connection:

1. In the IDE Connections panel of the Resources window, 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.

You can use connections in the Resources window 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 Applications window, under the Connections node.

To add a connection to an application:

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

Alternatively, drag the resource from the Resources window and drop it onto an application page.

Alternatively, drag the connection from IDE Connections in the Resources window and drop it onto the Application Resources panel in the Applications window.

Searching the Resources Window

You can initiate a simple search or an advanced search in the Resources window. You can also define a dynamic folder in a catalog so that the running of a query expression opens the folder.

There are two ways of searching in the Resources window:

- 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 Resources window, and how long it takes to connect to them, and the results are displayed in the Search Results panel.

How to Perform a Simple Search

Use the procedure to conduct a simple search, which spans across all the contents of the Resources window.

When you perform a simple search, the search is performed across all the contents of the Resources window, and it may take some time because JDeveloper connects to remote resources during the search.

To perform a simple search:

1. In the Resources window, 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 Resources window.
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.

How to Perform an Advanced Search

Use the procedure to start an advanced search to specify several search criteria, and select the location from which to start the search.

Alternatively, you can perform an advanced search where you specify a series of search criteria, and choose where to start the search from.

To perform an advanced search:

1. In the Resources window, 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 Resources window contents.
3. Enter search criteria to return the resources you want, and click **Search**.

How to Stop and Save a Search

You can stop a search before it is completed. Use the procedure to save search criteria by creating a dynamic folder, or save search results by creating a static folder.

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

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 Search Results panel of the Resources window, 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 Resources window.
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**.

Filtering Resources Window Contents

Use the procedure to filter the contents of My Catalogs so as to refine its contents.

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

To filter the contents of My Catalogs:

1. In the Resources window, 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.

Importing and Exporting Catalogs and Connections

Import Resource catalog archive files to share catalogs and connections. Use the procedure to export and import a catalog and thus facilitate the sharing of the catalog.

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 Resources window, 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 Resources window, 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.

Working with Resources Window Catalogs

Resources from multiple repositories are organized in a catalog either for individual developers or for specific groups. Resources in a catalog are organized in catalog folders that contain folders, dynamic folders, or filters.

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.

Creating Catalogs

Use the procedure to create a catalog. Organize information in catalogs in the Resources window.

You can organize the information in the Resources window in catalogs.

To create a catalog:

1. In the Resources window, 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.

Renaming Catalogs

Use the procedure to rename a catalog.

You can rename catalogs.

To rename a catalog:

1. In the Resources window, 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.

Working with Catalog Folders

You can create folders to organize the contents of catalogs.

How to Create Folders

Use the procedure to create catalog folders to organize the contents of a catalog.

You can organize the information within catalogs in folders.

To create a catalog folder:

1. In the Resources window, 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.

How to Create Dynamic Folders

A dynamic folder is defined by a query expression that opens the folder when it is executed. Use the procedure to create a dynamic folder:

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 Resources window, 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.

How to Add Resources to a Catalog

Use the procedure to add a resource to a catalog by means of a connection in the IDE Connections panel. Add resources by dragging an item or folder in a catalog to another folder in the same catalog, or to another catalog.

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

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.

To add a resource to a catalog:

1. In the Resources window, 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.

Working with Source Files

Learn about JDeveloper's editors for Java, XML, HTML/JSP, JavaScript, and PL/SQL.

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

Working with Index Data

JDeveloper stores information such as index or index data and the compiler `.cdi` files in the `.data` directories for the files that are used to build an application. Use the procedure to refresh index data.

When you develop an application in JDeveloper, the IDE stores information in the `.data` directories for the files used to build your application. JDeveloper stores many items in the `.data` directories, but the two most important are:

- a cache of information about the files that are part of your project; these are sometimes referred to as the index or index data.
- the compiler `.cdi` files, which contain information generated by the Java compiler.

While the compiler uses both of these, the IDE relies on the information in the index data for many other operations, such as to provide code assistance, auditing, refactoring, and many more. The information in the `.data` directories can be expensive to generate, and on complex applications can result in long compile times. Keeping the data in an easily available cache makes it more effective and results in faster operation while using the IDE.

As a project or application changes, however, the cached data may become out of date. You can update the index data for your project or application with the Refresh Client button. Updating the index data is context-sensitive, depending on both the content of the Projects panel and your selection in the Refresh Client menu.

To refresh the index data:

1. To update the index data for just the current project, select **Refresh Project**.
2. To update the index data for your entire application, select **Refresh Application**.

Using the Source Editor

JDeveloper provides editors, such as the Java Source Editor and the XML Editor, to edit source files. The editors provide features such as bookmarking, code insight, code templates, breadcrumb navigation, and overview popups for quick coding.

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
- PL/SQL Source 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.

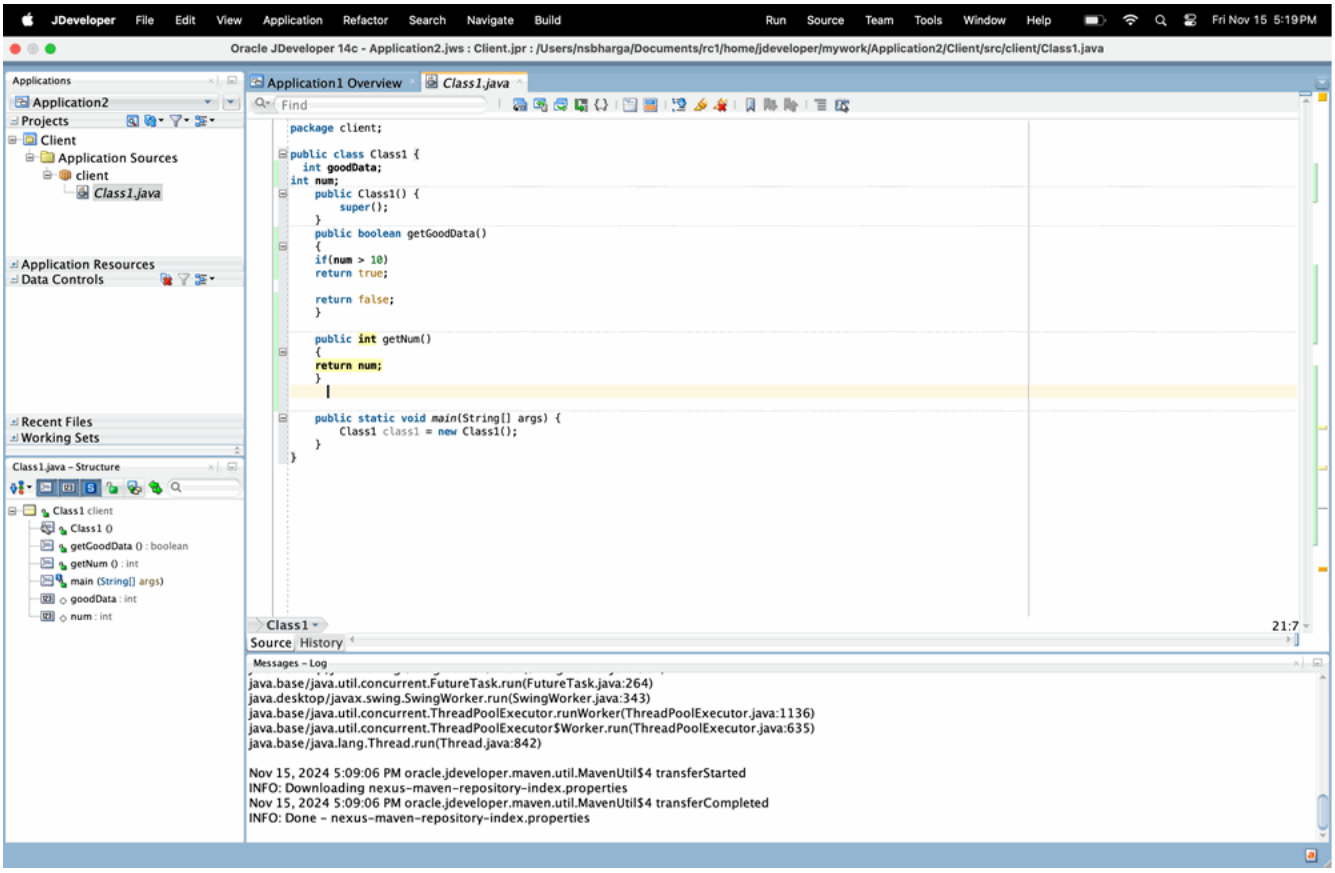
The following overview indicators are provided:

- 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

As shown in Figure 3-1, the right margin displays a miniature view of the source code. Press the Shift key and hover over this view to display code not currently viewable in the editor. By adjusting the position of the mouse while pressing Shift, you can view the entire code without scrolling in the editor itself.

Figure 3-1 Miniature View of Source Code



Overview Edit Marks

The overview strip in the right margin of the source editor shows an additional strip of marks indicating sections of the file that have changed. This allows you to instantly see sections of the file that have changed and quickly navigate between them.

You can use the following key to understand what the marks signify:

- Green mark: Addition
- Purple mark: Change
- Red mark: Deletion

A mark can be clicked on to instantly navigate to that point, or it can be hovered over to show a popup of the change.

As you work on your code, the overview strip on the right margin displays a solid line and a doted line. Solid indicates caret position. Doted indicates last edit. Hover over these indicators to view the referenced sections.

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 > Block 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.

Features Available From the Context Menu

The context menu of the generic source editor provides a set of features. These features include Completion insight which provides a list of possible completions; Parameter insight which provides the types and names of the parameters of the method being typed; and Complete Statement which provides auto-completion of code statements.

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.

Using Mini-Maps

The Mini-Map feature provides high-level views of the code structure, in the currently worked on file, in a size-adjustable strip alongside the code editor.

The Mini-Map offers a zoomed out 'live' view of the current file and displays it as a strip alongside the code editor. Changes in the source code are shown in the Mini-Map.

The Mini-Map allows you to see code structure at a higher level; a very useful feature when coding large files. Additionally, you can click on the Mini-Map to quickly navigate a source file.

The Mini-Map size can be adjusted via a right click menu and it offers a Google Maps like way to select views:

- **Satellite**—zoomed out text
- **Logical**—text is replaced by colored boxes indicating methods and class structure
- **Hybrid**—logical information is overlaid onto the satellite view

Using Stepping Margin

During application debugging, JDeveloper displays a stepping margin indicating the debugger location and relevant breakpoint data. The stepping margin may be hidden or displayed.

While debugging an application JDeveloper displays a stepping margin indicating the debugger location and relevant breakpoint data. You may select to hide the stepping margin by right clicking on the margin and deselecting the **Show Margin** option.

Using Multi Cursor

Use the Multi Cursor functionality in JDeveloper to type a text string and have it simultaneously populated on multiple lines. Use the procedure to use the Multi Cursor function.

The Multi Cursor function in JDeveloper enables you to type a text string and have it populated in multiple lines simultaneously. To use the Multi Cursor function:

1. Open a Java class.
2. Place the cursor on any code line.
3. Go to **Edit** > **Multi-Cursor**. Choose any of the three options: Add Cursor Above, Add Cursor Below, or Select Highlights. For example, you select the **Add Cursor Below** option.

   A second cursor displays on the line below.

   > **Note:**
   >
   > Option **Add Cursor Below** will be inactive if the cursor is placed at the bottom line of the code and similarly the **Add Cursor Above** will be inactive if the cursor is placed at the first line of the code.

4. Start typing.

   Your text appears in both lines. The following table describes the keyboard shortcuts to use the multi cursor functionality:

Table 3-28 Keyboard Navigations for Multi Cursor Functions

Navigation	Keys
Add Cursor Above	Alt+Shift+Page Up
Add Cursor Below	Alt+Shift+Page Down
Select Highlights	Alt+Shift+S

To return to the single cursor functionality, go to **Edit** > **Multi-Cursor** > **Clear Multiple Cursors**.

Using Hyperlinking with Javadoc Comments

Use the procedure to create hyperlinks in the Javadoc comments of your code.

JDeveloper enables you to create hyperlinks in the Javadoc comments of your code by following these steps:

1. Open a Java class.
2. From the main menu, click **Source** < **Add Javadoc Comments**.

   A comments section is created at the top of the code file.

3. Type a URL such as http://www.google.com
4. From the main menu, click **Tools** < **Preferences**.

The Preferences dialog opens.

5. From the hierarchical tree, click **Mouse Actions**.

   The **Mouse Actions** page is displayed.

6. Go to the **Activate Via Hover and...** column and on the Hyperlinks row select **alt** from the dropdown list. Click **OK**.

7. Hover the mouse over the URL while holding down the **alt** key and click the URL.

   A browser opens at http://www.google.com

Using the Find and Replace Toolbar

JDeveloper provides a code Find toolbar to find and replace whole words, regular expressions, and selected text. The toolbar is also used for more complex searches and other functions.

Table 3-29 Functions of the Find and Replace Toolbar

Function	Description
Match Case	Select to perform a case-sensitive search.
Whole Words	Select to search for whole words that match the string entered in the **Find** field
Regular Expressions	Select to treat the text in the **Find** field as a regular expression
Selected Text Only	Highlight a section of the code. The **Find** function will be executed over the highlighted code only.
Wrap Around	The found entries are highlighted on a different color.
Replace	When selected, a new bar is displayed below the **Find** bar. Use this option to find and replace data in your code files. You may replace a single entry, skip an entry, or replace all entries.
Find in Files	Select this option for more complex searches. When clicking this option a dialog is displayed with a help button. Click that button to find information relevant to that dialog.
Quick Outline	Displays an outline highlighting the Java methods to aid in finding the appropriate portions of the code to be searched.
Surround	Launches the **Surround** dialog. Click the question mark in that dialog to learn more about this function.
Generate Accessors	Launches the **Generate Accessor** dialog. Click on the dialog help to learn more about this function.
Override Methods	Launches the **Override Method**s dialog. Click on the dialog help to learn more about this function.
Implement Interface	Launches the **Implement Interface** dialog. Click on the dialog help to learn more about this function.
Reformat	Highlights a portion of the code for easy reformatting.
Bookmark	Inserts a bookmark on the found instance of the found entry. Once selected, additional options are displayed in the **Find** toolbar to **Go to Next Bookmark** or **Go to Previous Bookmark**.
Show Selected Element	Shows the found element and hides the rest of the code.
Show Block Coloring	Displays a doted line around blocks of code.

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.

How to Set Indentation Size for the Source Editor

Use the procedure to change the default indentation style in the source editor.

JDeveloper uses a default indentation style in the source editor. You can set your own indentation size based on your programming preferences.

To set indentation size 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 **Indentation Size**.
5. Change the indentation value as required.

> **Note:**
>
> While editing code, if you press the Tab key when the **Use Tab Character** option is unchecked, JDeveloper indents by the indentation size you specify (4 by default). If you select **Use Tab Character**, JDeveloper will use tab characters for indenting, based on values specified in both the **Indentation Size** and **Tab Size** fields. For example, if you use an indent size of 4, and a tab size of 8, then it takes two indent levels (4 spaces each) to reach the tab size (8). So if you press Tab twice to indent twice, JDeveloper will insert a tab character in the source file. That tab character will expand to 8 spaces.

6. Click **OK** to close the dialogs.

How to Set Fonts for the Source Editor

Use the procedure to set fonts, font type, and size for the source editor.

You can set fonts for the source editor, including font type and size.

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**.

How to Set Caret Behavior for the Source Editor

Use the procedure to set caret behavior, including blinking, blink rate, and caret shape for the source editor.

You can set caret behavior for the source editor, including blinking, blink rate, and caret shape.

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**.

How to Set Display Options for the Source Editor

Use the procedure to set options for general display features, such as breadcrumbs, scroll tips, and the right margin, in the source editor.

You can set options for general display features in the source editor, including options for breadcrumbs, scroll tips, the right margin, and brace matching.

To set display 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 **Display** node.
4. On the Display page, enter the settings you wish for the right margin.
5. Click **OK**.

How to Set Line Gutter Behavior for the Source Editor

Use the procedure to set the line gutter behavior, including colors, line selection, and line number visibility for the source editor.

You can set the appearance of line gutters for the source editor. JDeveloper allows you to specify colors, line selection, and line number visibility.

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**.

How to use the Save as HTML parameter in the Source Editor

Use the procedure to save Java source files in the HTML format.

JDeveloper allows you to save Java source files in HTML format via the **File > Save** as HTML command.

To customize how JDeveloper tags your HTML 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, expand the **Code Editor** node.
3. Select the **Save As HTML** node.
4. On the Save As HTML page, customize the HTML markup by selecting the available options
5. Click **OK**.

How to Set Options for Syntax Highlighting in the Source Editor

Use the procedure to set the options for syntax highlighting in the source editor.

You can control the colors and font style used by the source editor.

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**.

How to Set Bookmark Options for the Source Editor

Use the procedure to set bookmark options for the source editor. Specify the persistence and search behavior of bookmarks that you create in JDeveloper.

You can specify the persistence and search behavior of bookmarks you create in JDeveloper. On the Bookmarks page, decide how you want to handle bookmarks once you've exited the editor or Oracle JDeveloper, how to traverse bookmarks, and how to handle bookmarks at the end of files for lines that may no longer exist

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, set bookmark options.
5. Click **OK**.

How to Customize Code Templates for the Source Editor

Code templates make for quick and efficient coding. Use the procedure to view existing code templates so as to edit the existing templates or create new templates.

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**.

How to Manage Source Files in the Editor Window

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

How to Change the View of a File

Follow the steps to maximize the view of a file in the source editor and to revert to its original size.

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 Applications window.

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.

How to Navigate Between Open Files in the Editor Window

Follow the steps to navigate through open files by tab, navigate through by order of access, or navigate using file shortcut key assignment.

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.

How to Display the List of All Currently Open Files

You can get the alphabetical list of all the files open in a given tab group displayed; get the alphabetical list of all the files open in the editor window displayed; or switch focus to a file.

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 (documents) currently open in the editor window, regardless of split or detached files:

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

To switch focus to a file (document) currently open in the editor window:

- From the main menu, choose **Window > Documents**. Select a document and click **Switch to Document**.

How to Access a Recently Opened File

Use the procedure to access a recently-edited file that was opened directly. Upon opening an application, JDeveloper reopens the files in the same state that they were in when the IDE was closed.

Oracle JDeveloper remembers the last files you have edited.

To access a recently-edited file:

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 Applications window, 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**.

When you close an application in JDeveloper with any files open, reopening the application opens the files in the same state they were in at closing.

How to Manage Multiple Editors for a File

Use the procedure to split the editor window horizontally or vertically and open a single file in multiple views. The file can be split into multiple views, and the views are automatically synchronized.

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. Right-click the file title and choose **Split Horizontally** or **Split Vertically**.

   The editor window is now split into 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:

- Right-click the file title and choose **Unsplit**.

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

How to Work With Multiple Files

When the editor window is split into several independent views, you can view multiple files simultaneously in independent windows. Use the procedure to move a file to a different tab group.

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:

- Right-click a tab in the editor and choose **New Document 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 multiple views back into one:

- Right-click a tab in the editor and choose **Collapse Document Tab Group**.

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.

How to Quickly Close Files in the Editor Window

In JDeveloper, you can close any file open in the editor window with a single click, close all files, close all files except one, close multiple files at once, or close selected files.

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 > Documents**.
2. In the Documents dialog, select the files to be closed and click **Close Document(s)**.

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.

Working with Mouseover Popups

JDeveloper provides popups that open with cursor movements for the Structure window and for text in an editor, and provide information on Javadoc, source code data values for debugging and breakpoints. Smart-Popups, more appropriate for specific locations, are activated by keystrokes.

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.

How to Locate a Source Node in a Window such as the Applications Window, Databases Window, Applications Server Window

Use the procedure to locate the node for a file that is opened in the editor.

You can quickly locate the source node in the Applications window 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 Applications Window.

How to Set Bookmarks in Source Files

Bookmarks help users locate relevant code quickly. Use the procedure to set or remove a bookmark in a source file.

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**.

How to Edit Source Files

Oracle JDeveloper provides several features for editing source files.

How to Open Source Files in the Source Editor

The source editor provides for quick and efficient coding. Use the procedure to open the source code in its default 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 Applications window, 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 Applications window, 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.

How to Edit Source Code with an External Editor

Use the procedure to edit a source code file that is open in JDeveloper with an external editor. Save any changes that are made in JDeveloper, and reload the externally modified files to persist all the changes to the file.

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 File Is Unmodified** option on the Environment page of the Preferences dialog.

> **Note:**
>
> You can also format Java code from the command line by invoking `ojformat.exe`, which is included in your JDeveloper installation.

How to Insert a Code Snippet from the Components Window into Source Files

Code snippets that have been added to the Components window can be added to files open in an editor. Use the procedure to insert a code snippet from the Components Window into a source file.

Once you have added code snippets to the Components window, 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 Components Window into a source file:

1. Open the file in the source editor.
2. If the Components window is not visible, open it by choosing **Window > Components**.

3. In the Components window 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 Components window, click the snippet name or icon.

The code snippet appears in the file.

How to Record and Play Back Macros in Source Files

Use the procedure to define the shortcut keys to record and play back macros in files that are open in the source editor. Use the other procedures to record a macro, and to play back the macro.

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.

How to Create Tasks

JDeveloper provides the preconfigured task tags `TODO`, `TASK`, and `FIXME` for tasks that are directly related to code in source files or tasks that pertain to applications, projects or general files. Use the procedure to create your own task tags.

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` preconfigured, and you can add your own task tags in the Tasks page of the Preferences dialog.

To add your own task tags:

1. From the main menu, choose **Tools > Preferences**.
2. In the Preferences dialog, select the **Task Tags** node.
3. On the Task Tags page, alter the 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. Within the source code file, create a comment line starting with // and one of the task tags, for example `//TODO`.
2. Continue to type the comment, which will at the same time appear as an item in the Issues window.

How to Compare Source Files

Source files within the same project or external projects can be compared. Use the procedures to compare a file currently being edited with its saved version, to compare one file with any other file on the disk, to compare one file with another file outside the project, and to compare two files within the same project.

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 any other file on the disk:

1. Place the focus on the current version open in the editor.
2. From the main menu, choose **File > Compare With > File on Disk**.

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

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 Applications window, 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**.

How to Revert to the Last Saved Version of a File

You can revert to the last saved version of the file you are editing, but the reversion undoes the changes made since the file was last saved. Use the procedure 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. Any changes you have made since the last save are undone.

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**.

How to Search Source Files

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

How to Search Text in an Open Source File

Use the procedure to search the contents of a source file that is open in the source editor, with the option to replace text.

You can search for text with the option of replacing it across your source file.

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 toolbar, enter the text string.

   Text previously searched for in this session of JDeveloper appears in the dropdown located next to the Find magnifying glass icon.
5. Select other search parameters accordingly.

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

You may invoke the Replace function by choosing **Search > Replace**. Alternatively, press Ctrl+R.

How to Search for a Single Text String

Use the procedure to conduct a simple search in an open source file for a single text string.

You can search for a single text string in your source file.

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.

How to Search All Files in a Project or Application

Use the procedure to search all files in a project or an application.

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**.

How to Print Source Files

Use the procedure to print a source file using the **Print** option for **File**.

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 Applications window.
2. From the main menu, choose **File > Print**. Alternatively, to preview printed output, select **File > Print Preview**.
3. In the Print dialog, select your print options.
4. Click **OK**.

Reference: Regular Search Expressions

Use characters called regular expressions to customize a search string through pattern matching. Regular expressions allow you to match a string against a pattern, or extract parts of the match.

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 regular expressions package, java.util.regex. For more information, see "Regular Expressions and the Java Programming Language" at http://docs.oracle.com/javase/tutorial/essential/regex/.

Working with Extensions

Many functions in JDeveloper are implemented as extensions, components that are integrated with the IDE. You can either add existing extensions to JDeveloper, or create new 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 Adding External Tools to 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.

How to Install Extensions with Check for Updates

Use the procedure to install extensions using the Check for Updates wizard.

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 by selecting **Tools > Features > Check for Updates**.

How to Install Extensions from the Provider's Web Site

If the Check for Updates wizard informs you of the preference of the extension provider that you download extensions directly from their web site, follow the instructions on the web site of the extension provider to download and install the extension.

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.

How to Install Extensions Directly from OTN

Extensions such as JUnit Extension, SQL*Plus Extension, Oracle Business Intelligence Beans, and other extensions to JDeveloper contributed by the JDeveloper community can be downloaded from the JDeveloper Extensions Exchange website on OTN.

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

https://www.oracle.com/tools/downloads/jdeveloper-extensions-exchange.html

The available extensions include:

- 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 Introduction to Developing 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.

How to Install Extensions Using the JDeveloper dropins Directory

JDeveloper supports **watched directory**, a location where a user or script drops files that are automatically discovered by JDeveloper upon a restart. Use the procedure to install an extension using the 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.

Using the Online Help

Access conceptual and procedural information, context sensitive online help topics, and available documentation by means of JDeveloper online help at the Help Center.

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 window 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.

Using the Help Center

Use the Help Center to browse tables of content, locate topics in the contents list, and search for installed and online content.

The Help Center enables you to browse the table of contents, locate relevant topics in the Contents list, and do a full text search of installed and online content. It also provides a Favorites list for saving links to frequently referenced topics. The Help Center comprises two windows: the window that displays either Contents or Favorites 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 3-30 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.
	Windows	Choose to display either the Contents list or the Favorites list.

Table 3-30 (Cont.) Help Center Toolbar Icons

Icon	Name	Description
	JDeveloper Forum	Launches an external browser instance and visits 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.
- **Search results** - Displays the results of the full text search.

How to Open the Online Help

The JDeveloper Help Center provides the help window and the help topic viewer to assist users. Use the listed methods to open online help.

The JDeveloper Help Center comprises two windows: the help window 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.

How to Search the Documentation

Initiate a full-text search of all online documentation, or search for information on Oracle Technology Network and the Fusion Middleware and Database Online Documentation

Libraries. Use the procedure to conduct a full-text search from the Help Center, or search from the main toolbar **Search** field.

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 **Search Options** 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 **Start search** 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.

How to Add Bookmarks to the Favorites Page

To make lookups easier, use the procedures to add or delete links to help topics on the Favorites page, or to create, rename, or delete a Favorites folder.

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**.

How to Customize the Online Help Display

Customize features of the Help Center window using the toolbars and the context menu.

You can customize some features of the Help Center window 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 **Windows** 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.

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

How to Open and Close Multiple Help Topics

Use the tabs and buttons on the UI to navigate through topics in the help system. Use the context menu to close one or more pages that are open in the help topic viewer.

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 **File 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 **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.

How to Print Help Topics

Print individual help topics or sets of topics. Use the procedure to print single topics or a container of 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.

Common Development Tools

Learn about JDeveloper's features and tools.

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

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.

Checklist

The Application Overview Checklist appears on the Application Overview pages, and provides a linear guide to configuring and building an application. Follow the steps to use the 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.

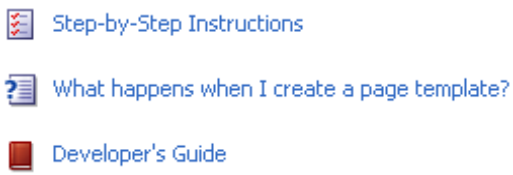
The checklist optionally walks you through the entire process of configuring and building your application, with links to specific dialogs and wizards. Each step is also designed to teach you about the architecture, tools and resulting files using a combination of links to step-by-step instructions, 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.

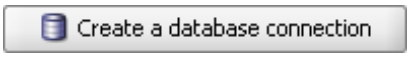
To use the Checklist:

1. Expand a step and read the prerequisites and assumptions.

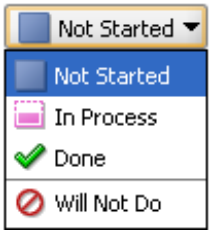
2. Optionally click any of the documentation links.



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



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



File Summary Pages

The Application Overview file summary pages display all the files and artifacts created for an application by object type that can be filtered by status and project. Use the File Summary Pages toolbar to add, edit, or delete objects.

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 3-31 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 3-32 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 editor.
	Delete	Removes the selected file or object.

Table 3-32 (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

File List

Use the File List to search for and work on objects that you have created within an application.

File List Tab Header

The file list tab header offers the Look in, Saved Searches, and Show History options to select a project. You can also save the criteria for searches, and reuse earlier searches.

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

Table 3-33 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.

Search Criteria Area

Use the search criteria area to input criteria for a search, determine the scope of the search, and save searches.

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

Table 3-34 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.
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.

Search Results Table

Use the options provided in the Search Results table to view the search results, rerun the search, list objects, or customize the table.

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

Table 3-35 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.

Compare Window

Use the numerous options, such as **Go to First Difference**, and **Go to Next Difference**, in the Compare Window to study the differences between two files or two directories.

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 Applications window, and with the Subversion source control system.

Toolbar

Use the Compare Window toolbar to view the differences between two files or two directories.

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

Table 3-36 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.

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.

Applications Window

The Applications window allows you to manage the contents and associated resources of an application.

Applications Window Toolbar

The Applications window toolbar provides the main list from which you can create or open an application. Use the Application Menu to open the New Gallery, to open a project, to deploy, close, delete, rename an application, or edit its properties.

This section describes the features available from the Applications window 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 3-37 Application Menu Options

Menu Option	Function
New Project	Opens the New Gallery ready for you to select the type of project to create.
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 Applications window.
Secure	Secures your application resources.
Deploy	Allows you to choose from the deployment profiles defined for the application.
Refactor	Allows you to choose from the refactoring options available for the application.
Compare With	Allows you to choose from the comparing options available for the application.
Application Properties	Opens the Application Properties dialog where you can set various properties for the application.

Application Operations

The Applications window allows operations such as move, size, float, minimize, maximize, restore or close the Applications window. You can also create or open an application using the window.

You can select several application operations from the Applications window. 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 window, or bring a currently open window to the foreground, using **Window > *window-name***.
- Dock, float, maximize, split (vertically or horizontally), restore or close the Applications window using the context menu available by right-clicking its tab or by pressing Alt+Minus.
- Change the application shown in the window 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).

Projects Panel Operations

From the projects panel of the Applications window you can perform operations such as rename and refresh the project, view its properties, and filter its contents.

You can perform the following operations from the projects panel of the Applications window:

- 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 window by selecting options from the **Applications Window 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 Applications window and then clicking Projects.

Application Resources Panel Operations

Use the Application Resources Panel to open or get the menu of commands for any node, the structure of an object, or search for objects. You can open and close the panel, resize it, or hide it.

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 Applications window 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.)

Data Controls Panel Operations

Use the Data Controls Panel to get a menu of commands for any node, edit the definition of a data control, or search for items in the panel. You can open and close the panel, resize it, or hide it.

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 Applications window 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.)

Recent Files Panel Operations

Use the Recent Files Panel to open an object in its default editor, or search for items in the panel. You can open and close the panel, resize it, or hide it.

You can perform the following operations in the Recent 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 Applications window and then clicking **Recent 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.)

Application Servers Window

Use the Application Servers window to create, edit, import, or export a connection to an application server. You can start the Integrated WebLogic Server, run it in the debug mode, create and update the Default Domain, diagnose problems, and launch the Admin Console for the Integrated and Oracle WebLogic servers.

The Application Servers window allows you to manage connections to application servers. It is integrated with the Resources window.

When you create an application server connection in the Application Servers window it is available in the Resources window. Similarly, when you create an application server connection in the Resources window, it is available in the Application Servers window.

From the context menu of the Application Servers window, 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 Servers window, 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 Servers window toolbar:

Table 3-38 Application Servers window 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.

Structure Window

The Structure window displays not only the code structure of an open file but also its UI structure. Multiple instances of the Structure window can be opened, and diagram objects can be dragged from the window and dropped directly onto diagrams.

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, which include the diagrams, the editors and viewers, and the Properties window.

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 Applications window, 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.

Structure Window Toolbar

The Structure Window Toolbar provides the Freeze and New View tools.

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

Table 3-39 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.

Structure Window Views

The Structure Window Views, such as the Cascading Style Sheet View and the Java View support the viewing and sorting of the structure of files based on the file type.

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 3-40 Structure Window Views

View	Description
ADF Business Components View	When you select any ADF business component in the Applications window, 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 Applications window or editor.
TopLink View	The TopLink view displays detailed information about the TopLink element selected in Applications window 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

Applications Window - Data Controls Panel

The Applications window - Data Controls Panel displays objects to which you can bind UI components. Drag and drop objects from the control panel onto an open web page or ADF Swing panel to create UI components.

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 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 3-41 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.

Log Window

Use the Log Window to view messages about functioning from the various components of the IDE. The contents of the log window can be selected, copied, saved, wrapped, or cleared.

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

The Log window displays information on:

- Apache Ant. When you build your project using Apache Ant, the Log Window displays relevant build information.
- Maven. When you build your project using Maven, the Log Window displays relevant messages.
- 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

Issues Window

Use Error, Warning, Incomplete, Info Task Marker, and Configure View in the Issues Window to view and resolve issues.

The Issues window enables you to view and manage application issues. It has the following features:

- It displays audit violations in file, project, working set, or application and provides information to help you resolve the issues. The Code Assist audit profile determines the audit violations that are reported.
- It displays a list of all warnings and errors encountered by the compiler after Make or Rebuild is executed.

  You can pin the information tab for a compiler operation and view the results of multiple Make or Rebuild operations by switching between tabs.
- It displays tasks specified in the source code.

  If you are working in a Java 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). These tasks then appear in the Issues window.
- Double-clicking on any item in the Issues window takes you to the corresponding source code.

- JDeveloper displays a series of tabs across the bottom of the Issues window, which highlight issues uncovered at different stages of testing and development (for example, when you build or run your application). If your application runs without issues, the tab displays a success message. If JDeveloper detects issues, the tab displays a description of the issue, the name of the file that caused the issue, the location of that file in the hierarchy of your application, and the name of the project containing that file.

The following table describes the Issues window toolbar.

Element	Description
Error	Toggle to show just errors in the selected scope.
Warning	Toggle to just show warning issues in the selected scope.
Incomplete	Toggle to show just incomplete issues in the selected scope.
Info	Toggle to show just the number of advisory issues in the selected file, or to list the advisory issues in the file.
Task Marker	Toggle to show just tasks.
Configure View Options	Toggle to configure the following view options: • **Table View**: Select to view items in a flat tabular structure. • **Tree View**: Select to view items in a nodal tree structure. In this view, items are sorted first by project, and then by file. • **Current Issues Scope**: Select to determine the scope of issues to be displayed. • **Preferences**: Select to open the Issues page of the Preferences dialog, where you can configure Issues window behavior.

Documents Dialog

Open the Documents Dialog from the Window menu to navigate between open documents.

The Documents dialog lets you navigate quickly among open documents in JDeveloper, including the Start Page.

At the bottom of the dialog, the Description panel lists the pathname to files that you have opened, in cases where you may have more than one file of the same name open. For example, if you have multiple files named `Class1.java` in different applications open at the same time, the pathname in the Description panel can help you identify which document is associated with which application.

To open the Documents dialog:

- Select **Window > Documents**.

JDeveloper displays a list of all documents currently open. You can select a document by selecting the document from the list, then clicking **Switch to Document**.

You can choose to sort how the documents are displayed by selecting an option under Order By:

- **Name** displays the documents in alphabetical order.
- **Recent Usage** displays the documents in the order in which they were most recently accessed.

Dependency Explorer

Open Explore Dependencies from the context menu of an element to understand the dependencies between project artifacts.

The Dependency Explorer provides information on the dependencies between project artifacts. You can open the Dependency Explorer from a node in the Applications window for an ADF application such as a `.jspx`, or from the node for a diagram.

To open the Dependency Explorer:

1. Right-click on the element you want to examine dependencies for in the Applications window.
2. From the context-menu, select **Explore Dependencies**.

   Alternatively, choose **Search > Explore Dependencies**.

For more information, click F1 from the Dependency Explorer window.

You can filter the types of file displayed by clicking the Filter button in the Dependency Explorer and choosing the types of file not include.

Adding External Tools to JDeveloper

You can extend JDeveloper by adding external tools.

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.

How to Find All External Programs Supported by JDeveloper

You can add those applications to JDeveloper that are recognized by JDeveloper. Use the procedure to find all external programs that JDeveloper is preconfigured to support.

You can identify any applications that JDeveloper already recognizes in order to add them to 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**.

How to Add Access to an External Program from JDeveloper

Use the procedure to add access to an external program from the JDeveloper IDE.

You can add access to an external program from the JDeveloper IDE.

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.

How to Change the Appearance of an External Program

Use the procedure to change how an external program appears, or remove access to an external program from JDeveloper.

You can change the settings for an external application, as well as remove an external application from the JDeveloper IDE.

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.

Working with Tasks

Register an issue tracker to be used as the task repository for a project. Then, use the Tasks window to create and save queries, create and organize tasks, and to find, update, and resolve tasks.

You can organize issues that are recorded on a registered issue tracker as tasks in JDeveloper. To work with tasks, you first need to specify the issue tracker that is used as the task repository for your project. After you register a task repository, you can use the Tasks window to perform the following:

- Find, update and resolve tasks
- Create new tasks
- Organize tasks by category
- Create and save queries

About Task Repositories

JDeveloper supports local and remote task repositories. Use local repositories to store personal tasks, and use remote task repositories to submit tasks, and assign responsibility for resolving tasks.

A task repository is a system for tracking issues that are submitted against a project. JDeveloper supports two types of task repositories:

- **Local** - JDeveloper includes a local task repository that you can use to store personal tasks. Tasks in your local repository are only stored on your local file system and are only accessible to you from within JDeveloper. You store scheduling details about tasks in your local repository.
- **Remote -** A remote task repository is generally located on a remote server and is accessible to other users. You can use a remote repository to submit tasks and assign responsibility for resolving tasks to members of a team collaborating on a project.

  Remote task repositories typically use an issue tracking system. JDeveloper provides support for the Bugzilla and JIRA issue tracking systems. For more details about the supported issue tracking systems, see the following sites:

- Bugzilla: http://www.bugzilla.org/
- Atlassian JIRA: http://www.atlassian.com/software/jira/overview

Working with Tasks

You can use the Tasks window to find, update and create tasks on a remote repository or your local repository.

Finding and Opening Tasks

Use id or a string to search for tags or create an advanced query on the Find Tasks page. Use the procedure to conduct a quick search of tasks.

From the Tasks window, you can perform a quick search for tasks by id or a string in the summary or open the Find Tasks page to create an advanced query. You can view a list of tasks that match your saved queries in the Tasks window.

After you save a query, the results of the search are listed under the query name in the Tasks window. You can double-click any task in the list to open the task form in a new window. You do not need to be online to open a task that is listed under a saved query. You can update a task in the and save the changes when you are offline and then submit the changes the next time that you are online.

To perform a quick search of tasks:

1. Choose **Window** > **Tasks** from the main menu to open the Tasks window.

   Alternatively, choose **Team** > **Find Tasks** to open the Find Tasks dialog and select the repository from the drop-down list. The drop-down list contains all remote task repositories that are registered with JDeveloper.
2. In the Repositories section of the Tasks window, click the **Search task in repository** icon for the repository that you want to search.
3. Enter a task id or string in the **Search task in Local Tasks repository** dialog.

   When you type in the text field, the dialog displays a list of possible matches that is based on tasks that you recently viewed.
4. Select a task from the drop-down list.

   You can choose **Search online Task Repository** in the drop-down list to retrieve more results.
5. Click **Open**.

   When you click **Open**, the task form opens in a window in JDeveloper.

Creating and Saving Task Queries

Use the Find Tasks dialog to create and save a task query, or use it to create a query as a URL. Use the procedure to create and save a task query.

You can save and name search queries that you use repeatedly. You create and save queries using the Find Tasks dialog or create a query as a URL. If you are not online, you can open tasks that are listed in the Tasks window. You can also update a task when you are not online and submit the changes later when you are online again.

To create and save a task query:

1. Choose **Window** > **Tasks** from the main menu to open the Tasks window.
2. In the **Repositories** section, click the **Create New Query** icon for the repository to open the Find Tasks dialog.

   Alternatively, you can choose **Team** > **Find Tasks** in the main menu to open the Find Tasks dialog.
3. Enter the search criteria.
4. Click **Search** to retrieve the results of the query.

   When you click **Search**, JDeveloper searches the remote repository and displays the search results in the form.
5. Click **Save Changes** in the search dialog.
6. Enter a name for the query and click **Save**.

After you save the query, the new named query is added under the repository node in the Tasks window.Expand the named query node to view a list of tasks that meet the search criteria. You can double-click any task in the list to open the task form in a new window.

You can open and update tasks in the list when you are not online. If you are not online, the **Submit Changes** button is disabled in the task dialog. You can click the **Save Changes** button to save any updates that you make to the task and submit the changes when you are online.

Reporting New Tasks

Use the local repository or use the Report a New Task dialog to create a new task in JDeveloper. Use the procedure to report new tasks.

To create a new task in JDeveloper, you need to use your local repository or register a remote task repository and then use the Report a New Task dialog to specify the details of the task. If you are not logged in to the task repository, you will be prompted to supply the log in details when you submit the new task. If you are not online, you can create and save the task and then submit the task when you are online.

To report new tasks:

1. Open the Tasks window and click the **Create New Task** icon for the repository in the **Repositories** section.

   Alternatively, choose **Team** > **Report Task** from the main menu and choose a repository in the **Report a New Task** form.
2. Enter the details of the new task.
3. Click **Finish** to submit a task to a remote repository or choose **File** > **Save** from the main menu to save a task to your local repository.

How to Add a Task Repository

Add a task repository so that you can use tools in JDeveloper to find, report, and resolve tasks. Use the procedure to add a task repository.

To use an issue tracker, you need to register it as a task repository. After the task repository is registered, you can use tools in JDeveloper to find, report and resolve tasks that are recorded in the task repository.

To add a task repository:

1. Choose **Window** > **Tasks** from the main menu to open the Tasks window.

2. Click the **Add Repository** icon in the **Repositories** section.
3. Enter connection details and click **OK**.

To modify the connection properties of a task repository:

1. Choose **Window** > **Tasks** from the main menu to open the Tasks window.
2. Right-click the repository node in the **Repositories** section and choose **Properties**.
3. Modify the connection details and click **OK**.

Working with the Tasks Window

The Tasks window displays a list of all tasks that are the results of a saved query. Use the window to create new categories and queries.

The Tasks window provides an organized overview of tasks that are recorded in a task repository. To use the Tasks window, you can use the local task repository or register a remote task repository with JDeveloper.

The Categories section of the Tasks window displays lists of tasks that are organized by category. The Repositories section of the Tasks window displays a list of all tasks that are the results of a saved query. You can create new categories and queries from the Tasks window.

Right-click a task in the window to view options such as opening, deleting, scheduling and assigning a category to the task. You can also move the mouse cursor over a task in the window to view a summary, including its status.

How to View Tasks

Organize tasks by assigning a task to a category, or save queries and view the results of the query using the Tasks window. Use the procedure to view tasks that are organized by category.

Use the Tasks window to view lists of organized tasks. You can organize tasks by assigning a task to a category. You can also save queries and view the results of the query in the Tasks window. You can double-click any task entry in a list to open the task in a new window.

You can enter a string in the **Filter** text field in the Tasks window to limit the tasks that are displayed to the tasks that contain the string in the task summary.

To view tasks that are organized by category:

1. Choose **Window** > **Tasks** from the main menu to open the Tasks window.
2. Expand a category node in the Categories section to see a list of tasks that you assigned to that category.
3. Double-click a task in the list to open the task in a window.

   You can move your cursor over a task entry to view a summary of the task.

To search for a task:

1. Choose **Window** > **Tasks** from the main menu to open the Tasks window.
2. Click the **Search Task in Repository** icon for the repository that you want to search.
3. In the **Search Task in Repository** dialog, enter the task id or part of the task summary text, or select from recent tasks in the drop-down list.
4. (Optional) Select a category from the drop-down list if you want to simultaneously assign the task to your category. Click **Open**.

When you click **Open**, the task opens in a window.

How to Organize Tasks

Use the Tasks window to create a custom category to which you can assign any task. Use the procedures to organize tasks by custom category, and to remove a task from a custom category.

You use custom categories to group tasks in the Tasks window. After you create a custom category, you can assign any task to that category and the task remains in that category until you explicitly remove it or you add it to a different category. A task can only be in one custom category. By default, the Categories section at the top of the Tasks window displays all tasks that are assigned to a custom category regardless of the status of the task.

The Categories section contains three default Schedule categories that group the tasks that have scheduling details (Today, This Week, All) and one default category for tasks that were opened recently. To hide a Schedule category, click the **Set Tasks window filter** icon at the top of the Tasks window and disable the category in the list. To hide the tasks that are resolved, click the icon and disable **Show finished tasks in categories**.

To organize tasks by custom category:

1. Choose **Window** > **Tasks** from the main menu to open the Tasks window.
2. Click the **Create Category** icon at the top of the Tasks window to open the New Category dialog box.
3. Enter a name for the category in the dialog box. Click **OK**.

   When you click **OK**, a node for the new category is added under the Categories section in the Tasks window.
4. Open a task in JDeveloper.
5. Right-click on the task and choose **Set category**.
6. Click **OK**.

   When you click **OK**, the task is added to the list of tasks under the category node.

You can also right-click a task entry in the Repositories section and choose **Set Category** to assign the task to a category.

To remove a task from a custom category:

1. Choose **Window** > **Tasks** from the main menu to open the Tasks window.
2. Expand a category node in the Categories section to see the list of tasks that are grouped in that category.
3. Right-click the task entry that you want to remove and choose **Remove from Category**.

You can remove a task from a category by assigning the task to a different category.

To remove all completed tasks from a category, click the **Remove all finished tasks from categories** icon.

4

Getting Started with 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. This chapter provides an overview of the features for developing applications available in JDeveloper.
This chapter includes the following sections:

- About Developing Applications with Oracle JDeveloper
- Creating Applications and Projects
- Managing Applications and Projects

About Developing Applications with Oracle JDeveloper

JDveloper provides different types of windows, visual and code editing tools, and testing and analytical tools for developing applications.

The features for developing applications in JDeveloper include:

- Different types of window 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 windows in the IDE, 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.

Creating Applications and Projects

An application heads the control structure view, while a project is positioned as the logical container for a program or part of a program. Create a new project, a new empty project, or open an external file into a new project to add a project to an application.

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 that defines 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.

All of the projects within an application are displayed below that application. If you filter a project using the working set, the project is not displayed under the application. Closing an application or project closes all open editors or viewers for files in that application or project and unloads the files from memory.

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.

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.

How to Create an Application

An application is the highest level view of objects in the control structure that tracks projects. Use the procedure to create a new application with a specific feature scope.

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..

To create a new application with a specific feature scope:

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**.

How to Create a Custom Application

You can either create an application with a specific feature, or create an application that can be customized to include any feature. Use the procedure to create a new custom application.

You can either create an application with a specific feature scope, or create a custom application that contains a single project that can be customized to include any features.

To create a new 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**.

How to Create a Project

A project is a logical container for a program or part of a program. Use the procedure to create a new project with a specific feature scope.

A project is a logical container for a set of files that defines a JDeveloper program or portion of a program. You can create a project that is preconfigured with a specific set of features.

To create a new project with a specific feature scope:

1. In the Applications window, 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 Applications window. It inherits whatever default properties you have already set. To alter project properties for this project, either double-click the filename or right-click and choose **Project Properties**.

Creating a New Custom Project

Use the procedure to create a new custom project.

To create a new custom project:

1. In the Applications window, 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**.

Managing Applications and Projects

Use the Applications window, which organizes projects to provide a logical view of the application, to manage applications and projects. View the contents within project folders in an application as a directory tree or a file list, or sort the nodes in packages and directories by type.

You can effectively manage your applications and projects using the Applications window. The Applications window organizes your projects in terms of higher-level logical components that gives you a logical view of your application and the data it contains. It provides an infrastructure that the different extensions can plug into and use to organize their data and

menus in a consistent, abstract manner. While the Applications window can contain individual files (such as Java source files), it is designed to consolidate complex data. Complex data types such as entity objects, UML diagrams, EJB, or web services appear in this Applications window as single nodes. The raw files that make up these abstract nodes appear in the Structure window.

Applications displayed in the Applications window contain one or more projects. Within the projects are the root folders for the paths in that project. You can choose to view project contents as a directory tree or file list, and packages by tree or list. And you can sort the nodes within packages and directories by type.

How to Open an Existing Application

You can either create a new application, or open an existing application. Use the procedure to open an existing application and add it to the Applications window.

You have the option of creating new applications from scratch or opening existing ones. As soon as you create or import the application, it is added to the Applications node in the Applications window.

To open an existing application and add it to the Applications window:

1. In the Applications window, 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 Applications window.

How to Open an Existing Project

A project that is created or imported is added to the selected application. Use the procedure to open an existing project and add it to an application.

As soon as you create or import a project, it is added to the selected application.

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

1. In the Applications window, 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.

How to Quickly Add Items to a Project Using the New Menu

Use the project context menu to add components to features for a newly created project. Use the procedure to add items using the project context menu.

After you have created your project, you can directly create components related to the project's features using the project context menu.

To add items using the project context menu:

1. Right-click the project you want to add items to and click **New**. Alternatively, you can click the **New** button in the toolbar after selecting your project in the Applications window.

> **Note:**
>
> The **File > New** menu displays the same items as the project context menu, and also contains additional application-level items.

2. In the New menu, click the item you want to add. If you want to choose an item from the New Gallery, click **From Gallery**. The second section of the context menu contains a list of recently created items for the current project type. The third section of the New menu contains a list of most-recently created items.

How to Import Existing Source Files into JDeveloper

Import a file into a project, or use it to create a new 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.

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

How to Import Existing Files into a New JDeveloper Project

Import a file into JDeveloper to create a new project. Use the procedure to open existing files, and import them 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 Applications window, 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.

How to Import a WAR File into a New JDeveloper Project

When you import a WAR file, JDeveloper creates a new project into which its contents are extracted. Use the procedure to open a WAR file, and import it 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 Applications window, 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.

How to Import an EAR File into a New JDeveloper Application

JDeveloper uses an imported EAR file to create a new application based on the extracted EAR modules. Use the procedure to open an EAR file, and import it into a new 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 Applications window 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 Applications window 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 Applications window, populated with projects based on the imported modules.

How to Import Files into a Project

Use an imported file in a project, or create a new project with an imported file. Use the procedure to open an existing file, and add it to a project using the Import Existing Sources wizard.

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.

> **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 Applications window, 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.

Managing 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.

How to Create a Folder or Java Package

Within projects, create custom folders or Java packages to organize project files. Use the procedure to create a folder or Java package.

To create a folder or Java package:

1. In the Applications window, 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.

How to Delete a Folder or Java Package

Use the procedure to delete a folder or Java package.

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.

How to Manage Working Sets

Configure the Applications window with Working sets to view a subset of files from the project. Use All Files to return to the default view.

Working sets allow you to configure the Applications window to show you a subset of files from your project. This is particularly useful when working with large projects. Until you create a working set, there are no default working sets. After you create a working set, the option All Files lets you to get back to the default view.

How to Group Objects into a Working Set

Run and debug a working set of a large application so that neither the whole application nor performance is affected. Use the procedure to group objects in the Applications window into a working set.

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.

To group objects in the Applications window into a working set:

1. In the Applications window, select the objects that you want to include in a new working set.
2. In the Applications window, 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**.

How to Create a Working Set by Defining File and Directory Filters

Select files or containers in the Applications window, or provide include and exclude filters in Manage Working Sets to define a working set. Use the procedure to create a working set by defining file and directory filters.

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

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

1. In the Applications window, 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.

How to Create a Working Set From Search Results in the Log Window

Exercise the option to save search results as a working set. Use the procedure to create a working set from the results of a search in the Log window.

You can create one or more sets of files for working on within a project. Based on the scope of your search, the action to save the search results as a working set is available.

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.

How to Identify the Current Working Set

Focusing the mouse pointer over Working Sets in the Applications window displays the name of the current working set as a tooltip.

A tooltip in JDeveloper identifies the working set you are currently using:

- In the Applications window, 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.

How to Change the Active Working Set

Use the procedure to edit files and projects in a working set.

To change the active working set:

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

  Files not belonging to the working set are removed from the Applications window but no editors are closed.

How to Edit Files and Projects in a Working Set

Use the procedure to edit files and projects in a working set.

To edit files and projects in a working set:

1. In the Applications window, 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.

How to Show All Files in the Applications window

Use the **All Files** option in **Working Sets** from the Applications window to show all files in the Applications window.

To restore the view in the Applications window to show all files:

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

How to Run and Debug a Working Set

A working set for debugging must include the projects that represent the Java EE modules. Use the procedure to run and debug a working set.

Before you begin, 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.

To run and debug a working set:

1. Choose **Tools > Preferences > Run**, then select the option **Always Run or Debug the current working set and its dependencies**.

> **Note:**
>
> This option is not available for the Database Developer or Java Developer roles.

2. Click **OK**.

   The next time you run or debug, JDeveloper will use the current working set. You can change to run and debug the entire application by choosing **Tools > Preferences > Run**, then selecting the option **Always Run or Debug the entire application**.

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

JDeveloper allows you to browse files without adding them to a project. Use the procedure 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 browse 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 Applications window 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**.

How to View an Archive

Open an archived file and view its contents, or add the contents to a new or existing project. Use the procedure to open an archive in JDeveloper and view its contents.

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 Applications window 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**.

How to View an Image File in JDeveloper

You can view image files in JDeveloper. Use the procedures to open and view an image in JDeveloper, and to view an image already imported into 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 Applications window 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 Applications window, select the image file.
2. Double-click the file, or right-click and choose **Open**.

Setting Default Project Properties

The project properties specified in the Project Properties dialog apply only to individual projects. The project properties specified in the Default Project Properties dialog apply to all projects across applications.

The project properties you specify in the Default Project Properties dialog apply to all subsequent projects you create across applications. Those you specify in the Project Properties dialog apply only to the current project. You can also set custom properties to override the properties set in a current project. This is particularly useful in a multiuser development environment. Custom properties are not stored in the .jpr file.

How to Set Default Project Properties

Overrides to the default project values apply only to individual projects. Use the procedure to view or change the default settings for a project.

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

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.

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**.

How to Set Properties for Individual Projects

Properties can be set for any individual project, or for all subsequent projects.

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

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

How to View or Change the Current Output Path for an Individual Project

Setting project properties for an individual project overrides the default project properties. Use the procedure to view or change the current output path for an 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 current output path for an individual project:

1. In the Applications window, 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**.

How to Set the Target Java SE for a Project

Specify the Java SE that JDeveloper will use when compiling and running a project. Use the procedure to view or change the current Java SE for an individual 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 Applications window, 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**.

How to Manage Project Dependencies

If a project that is related to another project through dependencies is compiled, the related project is also compiled. Use the procedure to manage the project dependencies for an individual project.

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 Applications window, 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**.

How to Associate Features with a Project Via a Template

Use a template in a project to avail the features available in the template. Use the procedure to associate features with a project by means of its template.

When you create a project from a template, the project will inherit all the features available in the template. More features are added dynamically to the project as you create content in your project. Go to **Project Properties > Features > Reconcile** to filter the list of features that are actually in use in the project.

To associate features with a project via its project template:

> **Note:**
>
> This procedure is applicable only for new projects created from a 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**.

How to Associate Features with an Individual Project

The JDeveloper design time filters the features to be associated with a project. Use the procedure to associate features with an individual 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 an individual project:

1. In the Applications window, 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**.

How to Set Javadoc Properties for a Project

Every JDeveloper project comes with defaults that you can replace. Use the procedure to set Javadoc properties for an individual 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 Applications window, 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.

How to Manage Libraries

Internal library definitions persist within the project file, cannot be shared, and are added using the Project Properties dialog. External libraries persist in stand-alone library definition files, can be shared, and their URLs too get added when they are added to a project.

There are two categories of library: internal and external. Internal library definitions are persisted within the project file itself (that is, within the `.jpr` file) and are thus always available to anyone opening that project. However, internal library definitions are not sharable among any other projects. You can add an internal library to a project by going to the Libraries and Classpath page of the Project Properties dialog, and clicking **Add JAR/Directory**.

External library definitions are persisted within their own stand-alone library definition file (that is, the .library file). As such, external libraries can be checked into source control, referenced by any number of projects, and can be shared among all users in a team environment in the same way that java source files are. Since external libraries have their own unique URL, adding an external library to a project adds that URL to the project.

How to Add Application-level Libraries and Classpaths

Libraries that are added at the application level are added to the libraries and classpaths of all projects within the application. Use the procedure to add an application-level library.

Libraries are often required across many or all projects within an application. All libraries that you add at the application level are implicitly added to the libraries and classpaths of all projects within the application and are readily available at design and compile time.

Any library or classpath added at the application level for use during deployment are included in the packaged EAR lib directory and is selected for deployment by default.

To add an application-level library:

1. Right-click the application in the application window.
2. Select **Application Properties** from the contextual menu.
3. Select the **Libraries and Classpath** node.
4. Click **Add Library**.
5. Locate the required library in the selection tree and click **OK**.

Libraries added at the application-level do not appear in the in the project properties Libraries and Classpath page by default. Check **Show Application Libraries** in the project properties Libraries and Classpath page to see all application-level libraries (this option is disabled if no application-level libraries exist). You can move the list of application-level libraries as a block to the top or bottom of the classpath entries using the Move to Top or Move to Bottom arrows. You cannot move an individual application-level library.

You can create and add an application-level library in the same way as described in How to Create a New Library and Add it to a New Project.

How to View the Current Libraries in a Project

Use the procedure to view the current libraries for an individual project. The classpath of a library is added to the project classpath when the library is added to the 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 Applications window, 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.

How to Add an Existing Library to a Project

Use the procedure to add an existing library to a project.

You can add an existing library to a project.

To add an existing library to a project:

1. With the project selected in the Applications window, 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**.

How to Create a New Library and Add it to a New Project

Use the procedure to create a new library, and add it to a project.

You can create a new library and add it to a new project.

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

1. With the project selected in the Applications window, 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**.

How to Edit an Existing Library in a Project

Use the procedure to edit an existing library in a project.

You can edit an existing library in a project.

To edit an existing library in a project:

1. With the project selected in the Applications window, 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**.

How to Remove Libraries from a Project

The classpath of a library is removed from the project classpath when the library is removed from the project. Use the procedure to remove a library 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 Applications window, 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**.

How to Import Libraries or Java SEs Outside the Project Scope

Work with libraries outside a JDeveloper project so that they are available to a project or a group of users. Use the procedure to work with libraries or Java SEs outside the scope of a project.

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**.

How to Create Libraries or Java SEs Outside the Project Scope

Create libraries outside the JDeveloper project so that they are available to a project or a group of users. Use the procedure to create libraries or Java SEs outside the scope of a project.

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**.

How to Edit Libraries or Java SEs Outside the Project Scope

Work with libraries outside the JDeveloper project so that they are available to a project or a group of users. Use the procedure to edit libraries or Java SEs outside the scope of a project.

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**.

How to Delete Libraries or Java SEs Outside the Project Scope

Work with libraries outside the JDeveloper project so that they are available to a project or a group of users. Use the procedure to delete libraries or Java SEs outside the scope of a project.

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**.

How to Manage Application and Project Templates

Use application and project templates to create applications with the specified combination of features.

Application and project templates provide you with a quick way to create the project structure for standard applications with the appropriate combination of features already specified. The new application created from template appears in the Applications window already partitioned into tiered projects, with the associated features set.

The application template you select determines the initial project structure, the named project folders within the application. The project templates define the associated features. You can then filter the work you do in JDeveloper such that the choices available are focused on the features you are working with.

How to Define a New Application Template

To standardize application development, use application templates that provides templates for projects. Use the procedure 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.

How to Define a New Project Template

Project templates provide standards for projects within an application. Use the procedure 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.

How to Share Application and Project Templates

Application or project templates are created in a location that can be accessed by other users. Use the procedure to create a shared template.

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.

How to Edit an Existing Application or Project Template

Users can edit templates to suit requirements. Use the procedure to edit an existing application or project template.

You can edit 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**.

How to Delete an Existing Application or Project Template

Application or project templates can be deleted. Use the procedure 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**.

How to Manage File Templates

You can create file templates with custom data for applications within JDeveloper. Use the procedures to create a template, and to edit an existing template.

JDeveloper allows you to create file templates with pre-populated custom data and use them in your application. For example, you can create a Java class file template that contains basic header tagging, then when you need to add a Java class file in your application, you would start from your created template and that would save you from manually entering basic header data in every single Java class.

You can create templates for all the files types supported in JDeveloper such as Java, html, xml, etc.

There are two workflows for managing file templates: you can edit an existing template and you can create a new template. The workflows are described below.

To edit an existing template:

1. Open your application in JDeveloper and navigate to **Tools > Preferences > File Templates.** JDeveloper provides an out-of-the-box Java class template.
2. In the File Templates dialog go to the folder directory and navigate to **General > Java.**
3. Click the provided template, **Copyrighted Class**. The template details and template content appears on the right pane.

   By default the built-in templates are not editable. If you start typing to modify the content of the template, you will be prompted to confirm that you want to modify the built-in template.
4. Modify the template by adding a simple line of text above the **@author** line. For example,

   *** Enter a description of this class.**

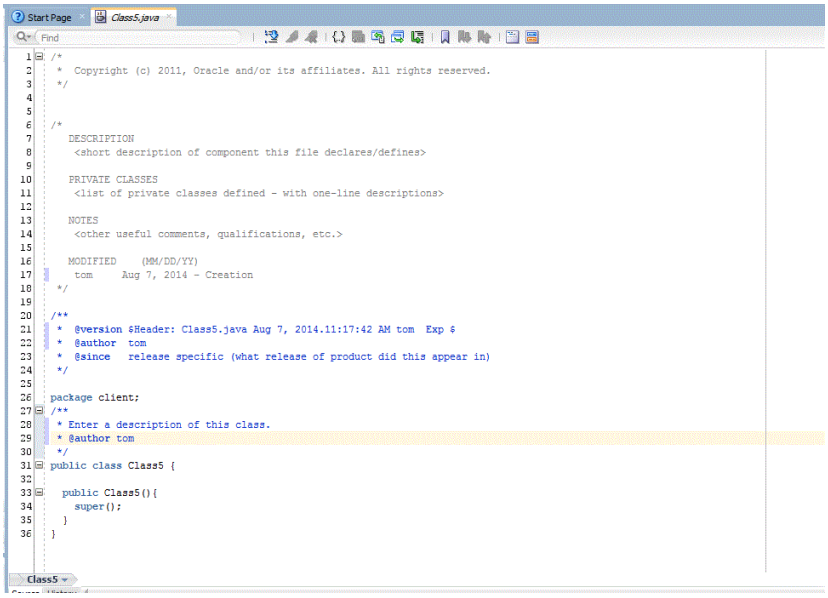
> **Note:**
>
> JDeveloper provides out of the box variables to use in your templates. To understand those variables see, Available File Template Variables.

You can use the **Reset** button on the top-right corner of the page to revert the template to its original text.

5. Click **OK** to close the Preferences dialog.
6. To use the template in your application, navigate to **New > Gallery > General > Java** and you will see on the right pane the **Copyrighted Class** template.
7. Select the template and click **OK**.

   After defining the File Name, the new class is created using the text in the created template.

Figure 4-1 Edit File Template



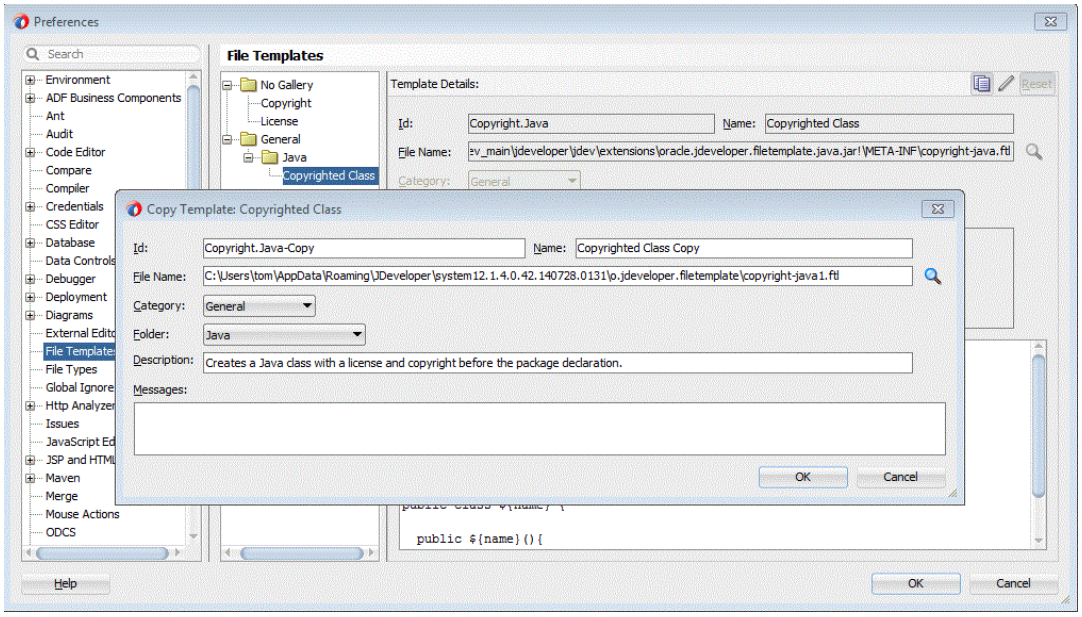
Line 28 shows the text that you added to the template on step 4.

To create a new template:

1. Open your application in JDeveloper and navigate to **Tools > Preferences > File Templates.** JDeveloper provides an out-of-the-box Java class template.
2. In the File Templates dialog navigate to the folder directory to **General > Java**.
3. Click the provided template, **Copyrighted Class**. The template details and template content appears on the right pane. You will use this Java class template as a starting point for developing your new template.
4. Click the **Copy** icon at the top right corner of the page.

   The **Copy Template - Copyright Class** dialog appears

Figure 4-2 New File Template Dialog



5. JDeveloper populates the new template details, but the following fields can be modified:
   - **Id**—the id for this new template
   - **Name**—he name of the template as it will appear in the Gallery
   - **File Name**—the path and file name for this new template. You can modify any part of the file name, including location.
   - **Category**—the main category in the Gallery
   - **Folder**—the id for this new template
   - **Description**—the description of the file as it will appear in the Gallery
   - **Messages**—the place where any errors in the template definition are described, for example if you enter a duplicate template id. The messages area is empty when there are no errors. This field cannot be modified.

   In this example we will only rename the new template as **My Copyrighted Java Class**
6. Click **OK**.

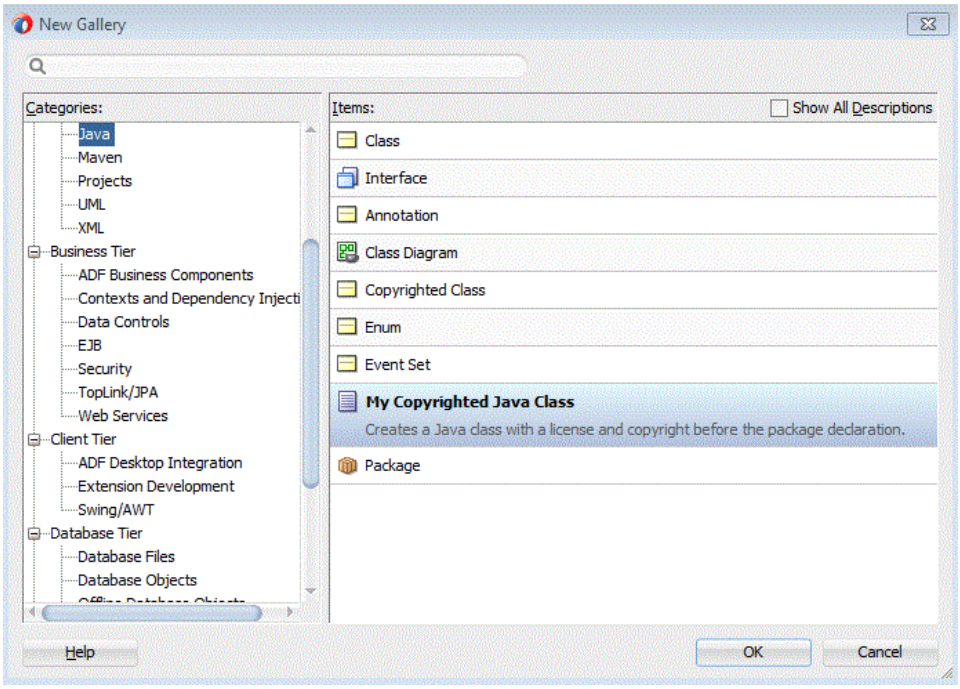
   A copy of the existing template is created.
7. Delete all the existing template text and enter the following:

```
/**
* Enter a description of this class.
 */

public class ${name} {

  public ${name}(){
    super();
  }
}
```

> **Note:**
>
> JDeveloper provides out of the box variables to use in your templates. To understand those variables see, Available File Template Variables.

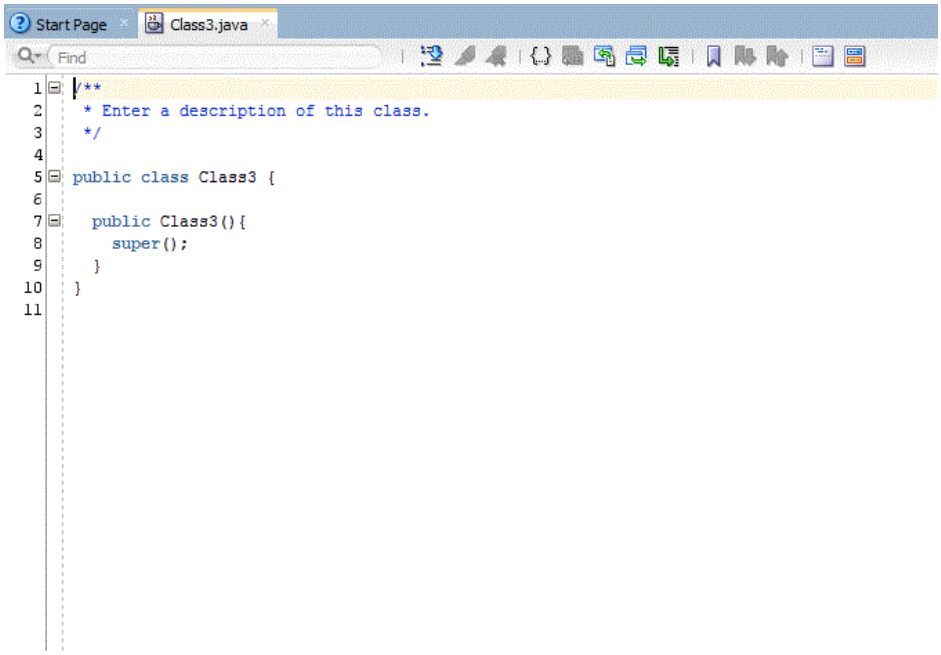
8. Click **OK**. The new template is created.
9. To use the new template in your application, navigate to **New > Gallery > General > Java** and you will see on the right pane the created **My Copyrighted Java Class** template.

Figure 4-3 New File Template - Select From Gallery



10. Select the new template and click **OK**. After defining the file name, the new class is created using the text in the created template.

Figure 4-4 New File Template - Editor View



Available File Template Variables

JDeveloper provides `name`, `nameAndExt`, `date`, `package`, `time`, `dateTime`, and `user` File Template Variables.

JDeveloper provides the following File Template Variables out-of-the-box:

- **name**—the name, without extension, of the file that will be created
- **nameAndExt**—the name, without extension, of the file that will be created
- **date**—the current time, in this format: Feb 16, 2008 from DateFormat.getDateInstance() during template processing
- **package**—the current package is specified
- **time**—the current time, in this format: 7:37:58 PM from DateFormat.getTimeInstance() during template processing.
- **dateTime**—the current date, from DateFormat.getDateTimeInstance() during template processing
- **user**—the current user name, as returned from System.getProperty("user.name")

How to Save an Application or Project

Use the **Save** or **Save All** options 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 an 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**.

How to Save an Individual Component or File

Use the **Save** or **Save As** on a selected application or project node to save the node. Use the procedure to save an individual component or file.

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 an 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.

To save an individual component or file:

1. In the Applications window, 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.

How to Rename an Application, Project, or Individual Component

You can rename application files. Use the procedure to rename an application or project container, or an individual source file.

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 Applications window, 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 Applications window 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**.

How to Relocate an Application, Project, or Project Contents

Files associated with projects or projects with applications can be moved using the Applications window. Use the procedures to move an individual file, an entire application or directory, or a group of files from one project to another.

The Applications window 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 Applications window, 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 Applications window, 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 Applications window 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 Applications window, 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 Applications window, 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 Applications window wherever you targeted them in step 4.

How to Close an Application, Project, or Other File

Any application, project, or file that is closed in the Applications window is unloaded from memory. Use the procedure to close an application or project.

When you close an application, project, or file in the Applications window, that application or project is unloaded from memory. When an application or project is closed, it appears in its unexpanded form in the Applications window.

In addition, you can remove applications, projects, or files from the Applications window, 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 Applications window, 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 Applications window 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.

How to Remove a File from a Project

You can either remove files from a project, or you can delete them permanently. Use the procedure to remove a file from a project.

You can remove files from a project, which removes them only from the Applications window list, or you can delete them permanently, wherever they reside, from within JDeveloper.

To remove a file from a project:

1. In the Applications window, 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**.

How to Remove a Project from an Application

Delete the project control file to remove a project from an application. Use the procedure 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 Applications window, 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**.

How to Remove an Application

Use the procedure to remove an application from the IDE.

You can close an application and remove it from the Applications window. If you want to delete the application

To remove an application from the IDE:

1. In the Applications window, click the **Application Menu**.
2. Select **Close Application**.
3. The Confirm Close Application Dialog is displayed. Select an option based on your preference.
4. Select **Application > Delete** to delete the application and its contents.

5

Developing Applications Using Modeling

This chapter describes how to use the modeling tools and technologies to create Unified Modeling Language (UML) class, profile, activity, sequence, and use case diagrams, as well as database, EJB, and business component diagrams to model your various business services and database structures.
This chapter includes the following sections:

- About Modeling with Diagrams
- Creating_ Using_ and Managing Diagrams
- Using UML
- Using Transformations
- Modeling with UML Class Diagrams
- Modeling with Activity Diagrams
- Modeling with Sequence Diagrams
- Modeling with Use Case Diagrams
- Modeling with Profile Diagrams
- Modeling with Java Class Diagrams
- Modeling with EJB Diagrams
- Modeling with Database Diagrams

About Modeling with Diagrams

Create the UML diagrams type and business services diagrams that are supported by JDeveloper using the New Gallery wizards.

JDeveloper supports six standard UML diagrams types, and four business services diagram types to model the software and systems development for your applications.

All of the diagram types can be created using the New Gallery wizards and are supported with the JDeveloper diagram editor, Components window, and the Properties window.

UML Diagrams

JDeveloper supports standard UML diagram types. The supported types are Activity Diagrams, Activity Diagrams with Partitions, Class, Sequence, Use Case, and Profile diagrams.

JDeveloper offers six standard UML diagram types:

- **Activity Diagram**. Model system behavior as coordinated actions. You can use activity objects to model business processes, such as tasks for business goals, like shipping, or order processing.
- **Activity Diagram with Partitions**. Model a single activity, showing its partitions as vertical divisions (lanes).

- **Class Diagram**. Model the structure of your system. Use to inspect the architecture of existing classes, attributes, operations, associations, generalizations and interface realizations.
- **Sequence Diagram**. Model the traces of system behavior as a sequence of events. Sequence diagrams primarily show this as messages between objects ordered chronologically.
- **Use Case Diagram**. Model what a system is supposed to do. A use case diagram is a collection of actors, use cases, and their communications.
- **Profile Diagram.** Define extensions to UML using profiles and stereotypes.

Business Services Diagrams

JDeveloper supports business services diagrams such as Business Components, Database, EJB Diagrams, and Java Class Diagrams.

There are four diagrams that support business services:

- **Business Components Diagram.** Model entity objects, view objects, application modules and the relationships between them.
- **Database Diagram**. Model your online and offline database tables and their relationships as well as views, materialized views, sequences, public and private synonyms.
- **EJB Diagram**. Model the entity beans, 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, enums, fields, methods, references, inheritance relationships, and implementation relationships.

Transformations

JDeveloper supports UML-Java Transformation, UML-Offline Databases Transformation, and UML-ADF Business Components Transformation. Transformation changes Java objects to UML classes, or UML classes to Java classes.

Transformation is the process of creating Java objects from UML classes, or creating UML classes from Java classes. The transformation types are as follows:

- **UML-Java Transformation**. See UML-Java Transformation.
- **UML-Offline Databases Transformation**. Transform a UML class diagram to an offline database and vice versa. See UML-Offline Database Transformation.
- **UML-ADF Business Components Transformation**. Transform UML classes to entity objects. See UML-ADF Business Components Transformation.

Creating, Using, and Managing Diagrams

Create a diagram, add components to it, and then either publish it as an image, or print it.

Oracle JDeveloper provides you with a wide range of tools and diagram choices to model your application systems. There are wizards to walk you through creating your diagrams and elements, as well as a Components window and Properties window to make it easy to drag and drop, and to edit a variety of elements without leaving your editing window.

The following images shows the diagram editor window, with a class diagram, as well as the Applications window and Components window. Open diagrams by double-clicking them in the Applications window, and once open, drag-and-drop components onto the diagram editor from the Components window.

Once you have created your diagram, add components to your diagram from the Components window. Zoom in and out of your diagrams with keystroke commands or view them at original size, or a percentage of the original size. When you are finished, you can publish your diagram as an image or print it using the right-click context menu or the main menu commands.

Figure 5-1 Create Class Diagram Example

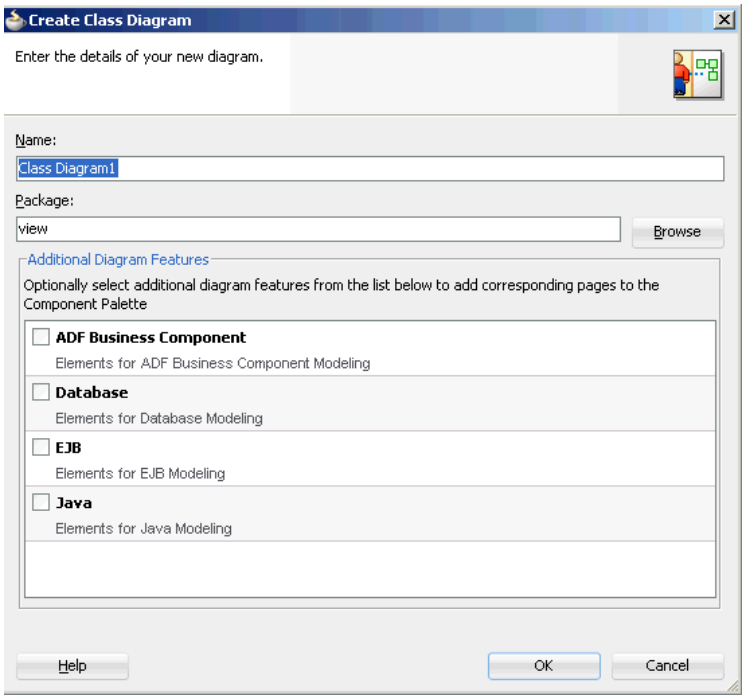
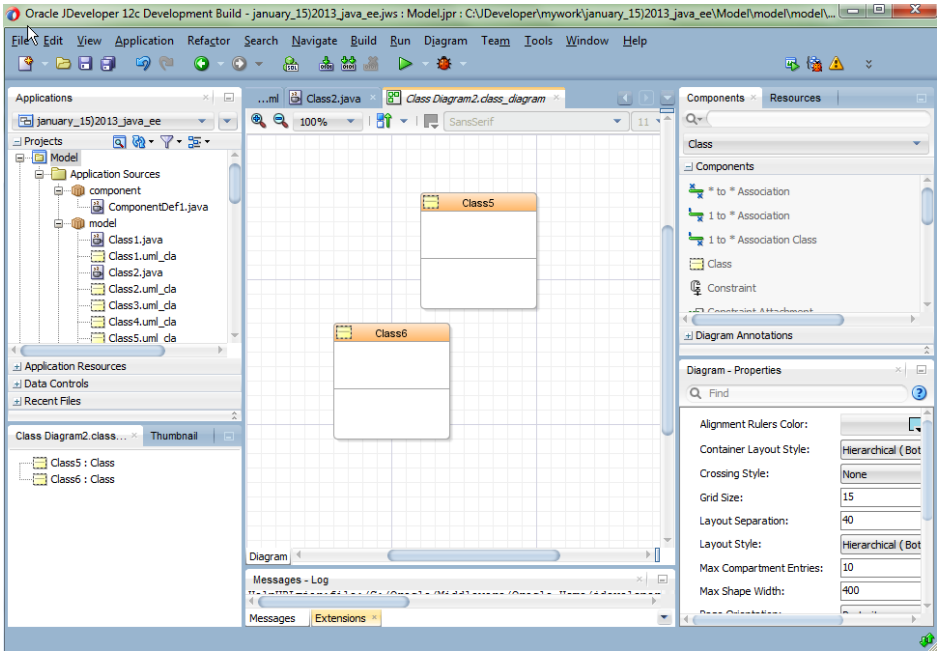


Figure 5-2 Class Diagram Example Showing Component Window



Creating a New Diagram

Use the procedure to create a new diagram. Use the New Gallery wizard to create a diagram to which components can be added.

The New Gallery wizard creates a diagram that is ready to contain your components. The wizard lets you choose the diagram type, specify the package, and select the components you want access in the Component window. Figure 5-3 shows an example create dialog for a class diagram.

To create a new diagram:

1. In the Applications window, select your project, then choose **File > New From Gallery** > **General** > **Diagrams**.
2. Select a diagram type, click **OK**.
3. 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**.

Working with Diagram Elements

Diagram elements can be dragged from the Components window on to diagrams.

Diagram elements are available in the Components window from which they can be dragged into the diagram. There are a variety of tools to help you manage your elements visually, as well as managing the properties of your elements.

How to Locate an Element on a Diagram

Clicking on the element name in the Structure window selects the element. Select **Thumbnail** from Window for a thumbnail of the diagram.

Click on the element name in the Structure window. The element is selected in the diagram. You can also use the thumbnail window of the diagram to find an element. To display a thumbnail view of a diagram, select the diagram either in the applications window or by clicking on the background of the diagram, then choose **Window > 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.

How to Select Specific Elements on a Diagram

Press and hold down the Ctrl key, then click the element on the diagram.

How to Select All Elements on a Diagram

Select all elements in a diagram to work on them simultaneously. The **Select All** option from Edit selects all elements.

Select all elements on a diagram to perform actions on all elements at the same time, such as align, copy, delete, or move. Click on the diagram surface, and then select any element, and choose **Edit** > **Select All**. You can also drag out an area on the diagram surface to select all or multiple elements.

How to Select All Elements of the Same Type

Use the **Select All This Type** option so that you can work with elements of the same type simultaneously. Use the procedure to select all elements of the same type.

If you want to edit or manage many elements of the same type, at the same time, use the select all option.

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**.

How to Deselect an Element in a Group of Selected Elements

Use the procedure to deselect a selected element in a group of selected elements. The deselect option removes an element from a group of selected elements.

If you select a group of elements, and you want to exclude particular elements, use the deselect option. This might be quicker than selecting the entire group one element at a time.

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.

How to Group Elements on a Diagram

Use the procedure to form a group of elements. For ease of working, group two or more elements and lock them in a container.

Grouping elements locks two or more elements into a purely visual container.

To form a group of elements:

1. Expand the diagram annotations accordion, if necessary. In the Components window, 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.

How to Manage Grouped Elements

Use the procedure to manage grouped elements on a diagram. Move elements and groups using the **Manage Group** feature.

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.

To manage grouped elements on a diagram:

1. Select the group to manage.
2. Right-click and select **Manage Group**.

   To move elements in and out of groups by Shift+drag the element to the desired position.

How to Change Semantic Properties

Use the **Properties** option to change the semantic properties of elements.

Open the Properties dialog in one of the following ways:

- Double-click an element on the diagram.
- Right-click an element on the diagram, and from its context menu, choose **Properties**.

How to Change Element Properties Using the Properties window

Use the procedure to change element properties.

1. Select **Window > Properties**. The Properties window displays both visual and semantic properties.
2. Select the diagram element.
3. Select the property to change.
4. On the right of the Properties window, select the control and change the value. (The control may be an edit box, a drop-down list, a checkbox, etcetera). If a single element is selected all valid properties are available; it multiple elements are selected, only the properties valid for all elements are selected.

How to Change the Element Color or Font

Use the **Visual Properties** option or the **Properties** option from Window to change element properties. Use the procedure to change the color or font of an element.

1. Select the element or elements on the diagram.
2. Then in the Properties window **(Window >Properties**), on the Graphical Options tab, select the current color (or the box for font type), make the required change(s), then press Enter.

   Alternatively, on the tool bar, select the font type, font size, or color box, then make the required change.

   Another option is to choose **Visual Properties** from the context menu, then make the required change.

How to Change the Visual Properties of New Diagram Elements

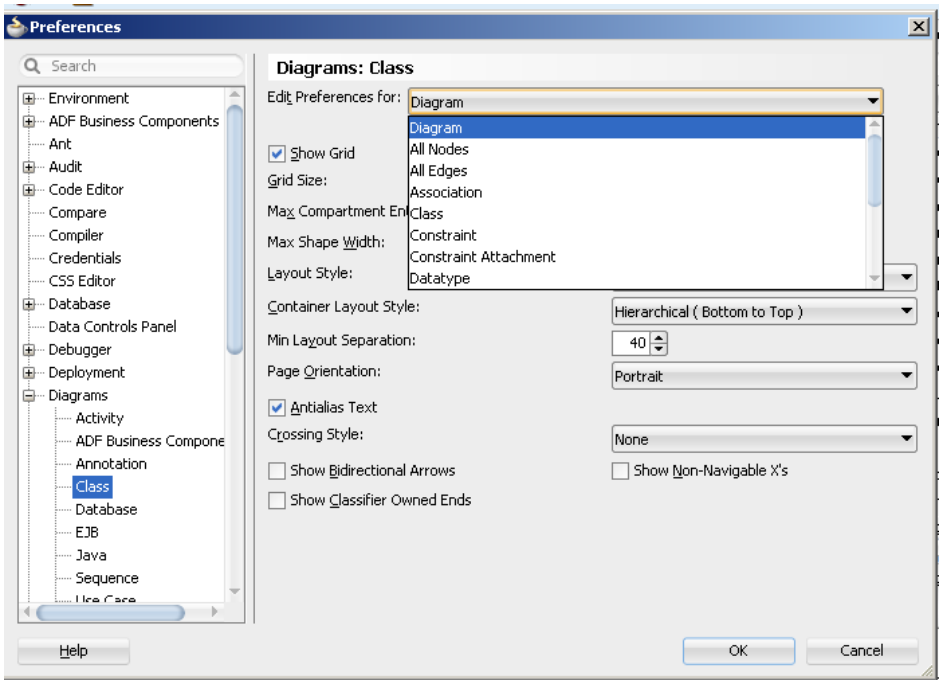
Use the procedure to change the default setting of elements to be added to a diagram. Define the default visual properties of any element that is added to a diagram using the Preferences dialog.

Use the Preferences dialog to define the default visual properties of any elements you add to your diagrams.

To change the default setting 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 5-3.The preferences might be displayed on a single panel or multiple tabs.
2. Change the default values as you require, and click OK.

Figure 5-3 Class Diagram Preferences Dialog



How to Copy Visual Properties

Use the procedure to copy and paste visual properties to elements. Copy preferences between elements using the Preferences dialog.

Use the Preferences dialog to copy preferences between elements.

To copy and paste visual properties to elements:

1. Select a diagram element.
2. Right-click and select **Copy Graphical options** from the context menu.
3. Select the target element(s).
4. Right-click and select **Paste Graphical Options** from the context menu.

How to Resize Elements on a Diagram

Use the procedure to resize a diagram element. Other than the initial and final nodes, you can resize any element to the required size.

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 nodes.

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 towards the visible edge of the diagram, the diagram is automatically scrolled. New diagram pages are added where an element is resized off the diagram surface.

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.

How to Display Related Classes on a Diagram

Extended, implemented, or referenced classes or interfaces can be included in a diagram. Use the **Related Elements** option to display related classes on a diagram.

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.

Choose from the following options to display related classes on a diagram:

- Select the class or interface, on the diagram, then choose **Diagram** > **Show** > **Related Elements**.
- Right-click the class or interface, on the diagram, then choose **Diagram** > **Show > Related Elements**.

How to Move Diagram Elements

Drag elements in to an element, or cut and paste elements to move them. Use the procedure to move diagram elements.

Dragging elements on the diagram surface is the easiest way of moving elements. To move elements over a larger areas, cut and paste. Whenever a diagram element is moved towards the visible edge of the diagram, the diagram is automatically scrolled. New diagram pages are added where an element is moved off the diagram surface.

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.

How to Undo the Last Action on a Diagram

Semantic property changes to an element can be reversed if the technology used is conducive to such actions. The **Undo** or **Redo** options undoes or redoes actions such as cut and paste, and also reverses changes to the font, color, and line width of diagram elements.

You can undo and redo your most recent graphical actions by choosing **Edit >Undo [...]** or clicking the undo icon. Graphical actions change the appearance of elements on the diagram surface and include the following:

- Cutting and pasting elements on diagrams.
- Altering the position and size of diagram elements.
- Changing the font, color, and line width of diagram elements.

Changes to an element's semantic properties can only be undone if the technology (Java, for example) permits it. Changing an element's semantic properties might prevent previous graphical changes from being undone.

To redo an action, choose **Edit > Redo [...]** or click the redo icon

How to Copy Elements to Another Diagram

The context menu or keystrokes can copy elements across diagrams. Use the procedure to copy elements from a diagram and paste them into another diagram.

Use the context menu or keystrokes to copy elements across different diagrams.

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.

How to Rename a Diagram

JDevelopr allows the renaming of diagrams. Use the procedure to rename a diagram.

Renaming changes the diagram name without leaving a copy of the original.

To rename a diagram:

1. In the Applications window, select the diagram to rename.
2. Choose **File > Rename**.

How to Publish a Diagram as an Image

The **Publish Diagram** option publishes a diagram as a graphic image. Use the procedure to publish a diagram as an image.

Use the right-click context option to publish your diagram as a graphic image. You can preview and print your diagram once it is published as an image.

To publish a diagram as an image:

1. Right-click your diagram, then choose **Publish Diagram**.

   Alternatively, click on the surface of the diagram, then choose **Diagram > Publish Diagram**.
2. Select the destination folder from the table for the image file.
3. From the File type drop-down list, select the file type for the image file (SVG, SVGZ, JPEG, or PNG).
4. In the File name box, enter a name for the image file, including the appropriate file extension (svg, svgz, jpg, or png).
5. Click **Save**.

How to Setup a Page for Printing

Use page setup to change the orientation or margins of a page before printing. Use the procedure to set up a page before printing.

Change from portrait to landscape or your margins for printing using page setup.

To setup 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.

How to Set the Area of a Diagram to Print

Use the procedure to set the area of the diagram to be printed. Use the print area option from File to define the area to print from a diagram.

Set a specific area of your diagram to print using set print area off the File menu option.

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.

How to See a Preview of Your Page Before Printing

Use the **Print Preview** in File to preview a page before printing. Use **Print Options** to add formatting, or to add content to headers and footers.

To see a preview of your page go to **File > Print Preview**. You can also set print options from this page by choosing **Print Options**. On the Print Option page you can add header and footer content as well as text formatting.

How to Clear a Diagram Print Area

In JDeveloper, diagrams come with the **Clear Print Area** option.

To clear a diagram print area choose **File > Print Area > Clear Print Area.**

How to Zoom in and Out of a Diagram

Use the JDeveloper zoom options to zoom in and out of diagrams.

Use **Ctrl+scroll** to zoom in and out of diagrams. When you are using the thumbnail view, use **scroll** to zoom. There are also zoom options on the diagram toolbar.

How to Display an Entire Diagram

Use the **Fit to Window** option for Diagram to view 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**.

How to Display the Selected Elements at the Maximum Size

The **Zoom to Selected** option for Diagram maximizes a diagram view.

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**.

How to Display a Diagram in its Original Size

The **100%** option for Diagram reverts the size of a diagram to its original size.

In the zoom drop-down list, located on the diagram toolbar, choose **100%**, or click the diagram, then choose **Diagram** > **Zoom** > **100%**.

How to Delete a Diagram

An unutilized diagram can be deleted using the menu bar. Use the procedure to delete a diagram.

Delete the diagram and related diagram elements using the menu bar.

To delete a diagram:

1. In the Applications window, 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 applications window and on the file system.

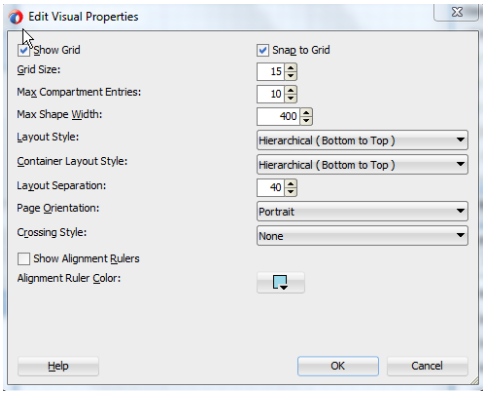
   You can also delete a diagram from the Applications window. In the Applications window, right-click on the diagram name and choose **Delete**.

Working with Diagram Layout

Use the Preferences option for Diagrams under Tools to design the layout of elements within diagrams.

Diagrams can be laid out in hierarchical, symmetrical, grid, and row styles. Elements within your diagrams can also have customized layout styles. There are many preferences available to customize the way your diagram looks. Most preferences can be set using the various diagram preferences dialogs at **Tools > Preferences > Diagrams** (diagram type), as shown in Figure 5-4. From the general preferences dialog you can choose **Edit Preferences for** to set specific preferences for new diagrams.

Figure 5-4 Class Diagram Visual Properties

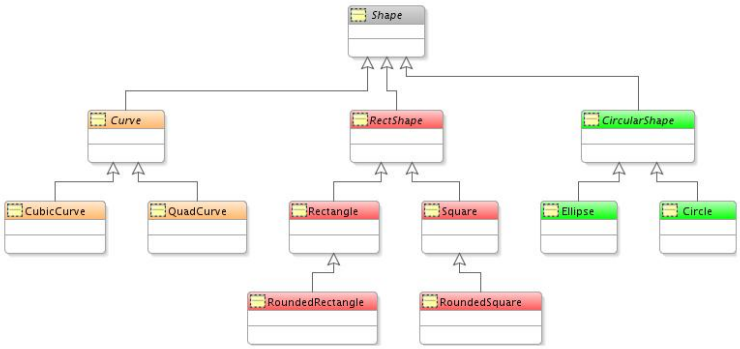


How to Use a Hierarchical Diagram Layout

Use the four orientations of hierarchical layout to structurally organize diagram elements.

Hierarchical layout puts diagram elements in hierarchies based on generalization structures and other edges with a defined direction, as show in Figure 5-5. Edges 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 5-5 Hierarchical Diagram Layout Example



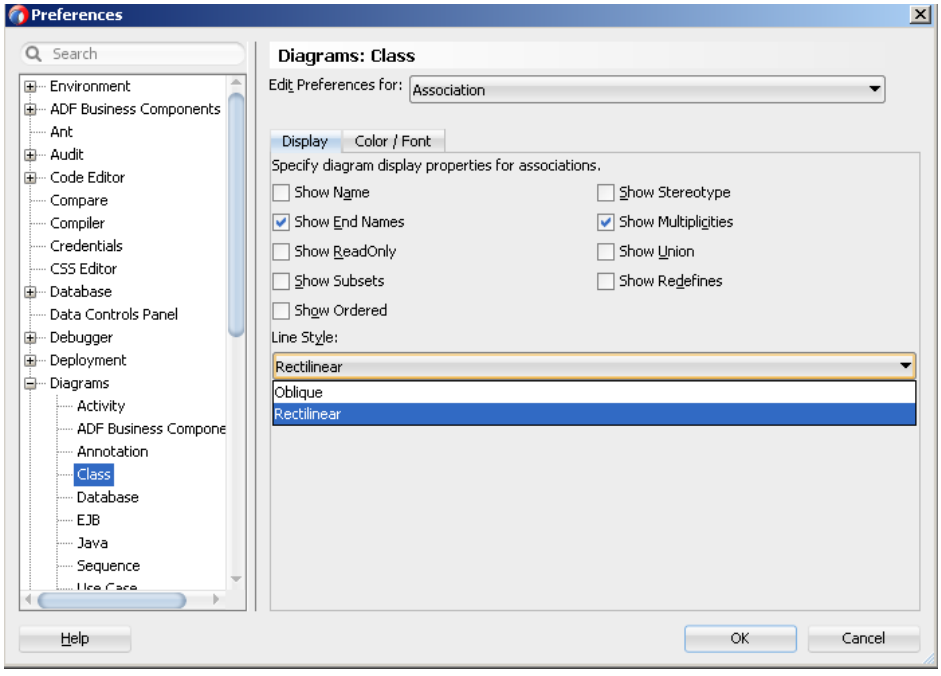
How to Use Layout Edges on a Diagram

Use oblique or rectilinear line styles for individual diagram edges, or for all instances of a diagram type. Use the Line Style preference under Tools or Edit to set the default line style for each diagram edge.

Diagram edges can be laid out in either oblique or rectilinear line styles. Oblique lines can be repositioned at any angle. Rectilinear lines are always shown as a series of right angles.

You can set the default line style for each diagram edge using the "Line Style" preference under **Tools > Preferences > Diagrams > Class** and from the **Edit Preferences** dropdown, select Association, as shown in Figure 5-6.

Figure 5-6 Diagram Preferences, Edit Preferences for Line Style



You can also set the line style for all instances of that diagram type. Keep in mind that when the line style of an individual edge is changed to oblique, and you change the line style from

rectilinear to oblique, no change will be apparent on the diagram, but you will then be able to move any of the lines on the diagram into a new position at any angle.

You can select individual diagram edges 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 reposition it (or portions of it) at any angle.

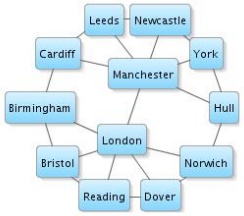
You can also choose the crossing styles for your lines to be bridge or tunnel style. Selecting bridge creates two parallel lines where the lines intersect. Selecting tunnel creates a semi-circle shape on the intersection. The default style is a regular crossing over of the two lines where the lines intersect.

How to Use a Symmetrical Diagram Layout

The edges between the nodes determine the symmetry of alignment between diagram elements. A symmetrical layout can arrange nodes in a radial layout around a central node, or in a grid layout.

Symmetrical layout aligns diagram elements symmetrically based on the edges between the nodes as shown in Figure 5-7. 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 5-7 Symmetrical Diagram Example

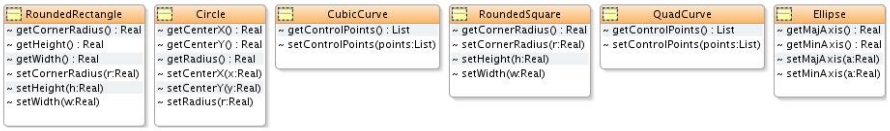


How to Use an Orthogonal Diagram Layout

Orthogonal diagrams have hierarchical and non-hierarchical elements arranged in a way that the aligned edges of a component follow the same direction.

Orthogonal diagrams show hierarchical and non-hierarchical elements where the aligned edges of a component all follow the same direction, as shown in Figure 5-8. For class diagrams, the generalization hierarchy is simply aligned, indicating the independence of each element.

Figure 5-8 Orthogonal Diagram Layout Example

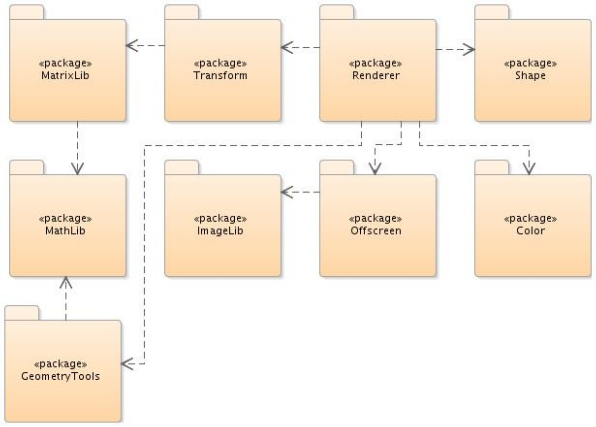


How to Use a Grid Diagram Layout

In grid layouts, diagram elements are positioned in grids with the nodes laid out in straight lines either in rows or in columns.

Grid layout puts 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, as shown in Figure 5-9. Nodes are laid out starting with the top left node.

Figure 5-9 Grid Diagram Layout Example



How to Lay Out Diagram Elements

Use Lay Out Shapes from the context menu of a diagram or the Layout Shapes option for Diagram to select layout styles for diagram elements. Use the procedure to lay out elements on a diagram.

Layout styles are available by opening the context menu for a diagram and choosing **Lay Out Shapes**, or by selecting the **Diagram > Layout Shapes**.

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.

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 a diagram, click on the diagram background and select **Visual Properties** from the context menu.

To set the default layout for new elements on a diagram go to **Tools > Preferences > Diagrams**.

How to Lay Out Diagrams Using the Grid

The preferences for Diagrams under Tools provides the Show Grid option to set the default diagram grid display and behavior, and the Snap to Grid option to snap elements to a grid.

Diagram elements can be automatically snapped to the nearest grid lines, even if the grid is not displayed on the diagram. Grid cells on the diagram are square and 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 set the default diagram grid display and behavior go to **Tools > Preferences**, select **Diagrams**. From there you can select the **Show Grid** checkbox to display the grid or **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.

To define diagram grid display and behavior for the current diagram:

1. Click the surface of the diagram.
2. In the **View > Properties window**, select to Show Grid, or Snap to Grid.

How to Distribute Diagram Elements

Distribute diagram elements so that elements are aligned to specific points. Use the procedure to distribute diagram elements.

Distributing diagram elements spatially arranges elements to specific point, such as top, bottom, etc. 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 distribute diagram elements:

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.**

How to Align Diagram Elements

Align elements vertically and horizontally, or space elements so that they are equidistant from each other. Use the procedure to align and size elements.

Elements can be aligned vertically and horizontally. You can also change the location of elements to have equal vertical and horizontal spacing.

To align and size elements:

1. Select two or more elements. 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** for 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** for all the selected elements to have the same height. The new element height is the average height of the selected elements.

4. Click **OK**.

Working with Diagram Nodes

A node in a UML diagram represents a physical device or an execution environment. Nodes can be created inside or outside elements.

In a UML diagram a node can represent a physical device or an execution environment. A physical device can be a single device or a configuration of multiple devices. An execution environment is a software container (such as an operating system or an EJB). You can create nodes inside or outside elements.

How to Create a Node on a Diagram

A node can be created using the Components window. Use the procedure to create a node on a diagram.

Create a node using the Components window.

To create a node on a diagram:

1. Select the node type you want to create from those listed in the Components window for your diagram.
2. Click the diagram where you want to create the node, or drag it from the component palette. This adds the node at its default size.

   Alternatively, 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.

How to Create Internal Nodes on a Diagram Element

Use the procedures to create an internal node on a diagram element, and to optimize the size of nodes on a diagram.

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.
- Relation usages.

Figure 5-10 Symbolic Diagram Class View

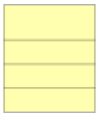
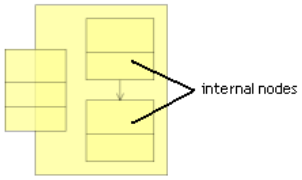


Figure 5-11 Expanded Diagram Class View Showing Internal Nodes



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 Applications window, or diagram, and drop it in the expanded node to create an inner node.

To change the way nodes are presented 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 optimize the size of nodes on a diagram:

Use the optimize feature of the right-click context menu to optimize your nodes. Optimizing will adjust the size of the nodes so that all attributes show.

1. Select the nodes to resize.
2. Right-click the selected nodes then choose **Optimize Shape Size > Height and Width**, as one option, or separately.

Working with Diagram Edges

Edges on a diagram may be hidden, either individually or in groups, and tagged 'hidden' in the Structure window. Diagram edges can be restored to view, either individually or in groups.

You can hide a single edge or any number of edges on your diagrams. Edges that are hidden on a diagram continue to show in the Structure window, with "hidden" appended. If there are any hidden edges, you can bring them back into view individually or all at once.

How to Hide Edges on a Diagram

Use the procedure to hide one or more edges on a diagram.

To hide one or more edges:

1. Select the diagram edge to hide. (To select all edges of a particular type, right-click an edge, then choose Select All This Type.)
2. Right-click and choose **Hide Selected Shapes**.

   You can also go to the Structure window, select the edge or edges to hide, right-click and choose **Hide Shapes.**

How to Show Hidden Edges on a Diagram

In the Structure window, select the edge to show, right-click and choose **Show Hidden Shapes**.

How to Show all Hidden Edges on a Diagram

In the Structure window, select the edges to show, right-click and choose **Show Hidden Shapes**.

How To List All Hidden Edges Together in the Structure Window

Right-click an object listed in the Structure window and choose **Order By Visibility.**

How to Change Crossing Styles on a Diagram

The default setting for crossing styles is no style. Changing the crossing style to bridge creates two parallel lines at the line intersection, and selecting the tunnel style creates a semi-circular shape at the intersection.

You can choose the crossing styles for your lines to be bridge or tunnel style. Selecting bridge creates two parallel lines where the lines intersect. Selecting tunnel creates a semi-circle shape on the intersection. The default style is a regular crossing over of the two lines where the lines intersect. You can also choose no crossing style, which is the default setting.

Annotating Your Diagrams

Explanations to diagram or diagram elements are included as notes. To add notes, use the options in the Components window.

Notes are used for adding explanatory text 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 Components window.

How to Add a Note to a Diagram

The Diagram Annotations feature in the Components window provides options for note addition. Use the procedure to add a note to a diagram.

Use the Diagram Annotations feature in the Components window to add a note to your diagram.

To add a note to a diagram:

1. Click the Note icon in the Diagram Annotations section of the Components window.
2. To create the note at the default size, click the diagram to create the note.

   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.

How to Attach a Note to an Element on a Diagram

Use the Attachment component of the Diagram Annotations feature in the Components window to attach notes to diagram elements. Use the procedure to attach a note to an element on a diagram.

The Components window Diagram Annotations feature provides an Attachment component to attach notes to your diagram elements.

To attach a note to an element on a diagram:

1. Click the Attachment icon in the Diagram Annotations section of the Components window.
2. Click the note.
3. Click the element that you want to attach the note to.

How to Change the Font on a Note

The note element provides options to make change to fonts in a note . Use the procedure to change the font format and size in a note.

Change the font using the standard editing options available on the note element.

To change the font size, color, bold, or italics on a note:

1. Click the note element. The text editing box appears.
2. Select the text to edit.
3. Select your text format.

Storing Diagrams

Diagrams are stored on disk as diagrams in a folder in the model path specified in the project settings. Use the step to set the model path.

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, which 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.

Using UML

Use UML element properties to customize display appearance and the semantic properties that determine element behavior. A UML application can be used to create diagrams and components.

The UML elements that are created independently in the New Gallery are listed in the application window and can be dropped onto your diagrams. You can also create a UML application, which allows you to quickly create diagrams and related components.

UML element properties allow you to customize both display appearance (graphical options such as color or font) or semantic properties which describe the behavior of the element when it is deployed (attributes, display options, class relationships, and so forth).

The general preferences dialog sets preferences for all diagrams of that type Right-click and choose **Visual Properties** to edit preferences for the diagram currently in your editing window.

Creating UML Elements Off a Diagram

Use the procedure to create UML elements off a diagram. New Gallery helps you create UML elements even without a diagram.

Use the New Gallery to create UML elements without a pre-existing diagram.

To create UML elements off a diagram:

1. Select the project in the Applications window.
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 applications window.

Storing UML Elements Locally

UML elements are stored in individual files in the appropriate package folder under the specified folder in the project model path. Set the model path using the Modelers option in Project Source Paths for Default Project Properties under Application.

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**.

Using UML Profiles

A UML Profile when applied to a UML model specifies additional semantics. JDeveloper provides two UML 2.4.1 standard profiles.

A UML Profile can be applied to a UML model to specify additional semantics. JDeveloper includes the two UML 2.4.1 standard profiles and a profile for transforming UML objects to Offline Database objects. It also allows other profiles to be registered. These may be third party or user defined.

How to Create a Profile

Use the procedure to create a UML profile.

Follow these steps to create a profile.

1. Create a new project.
2. Right-click the project and select Project Properties from the context menu. Select the Libraries and Classpath node. Click the **Add Library** button on the right.
3. In the Add Library dialog, select UML 2.4.1 Metamodel and click **OK**.
4. Right-click the project and select **New > From Gallery**.
5. In the General category, select the UML node. In the Items pane, select Profile, and click **OK**. The Properties editor opens.
   a. Give the profile a name. The name is used to identify the profile when it is applied.
   b. Give the profile a URI. This is the namespace URI that will be used for the XML namespace when persisting an applied profile.
   c. Make sure that the profile has no owning package.

   You can create both Stereotypes and Extensions as Package Elements within the Profile, however it is easier to do this using a Profile diagram.
6. Click **OK** to close the Project Properties dialog.

How to Export a Profile

Use the procedure to export a UML profile.

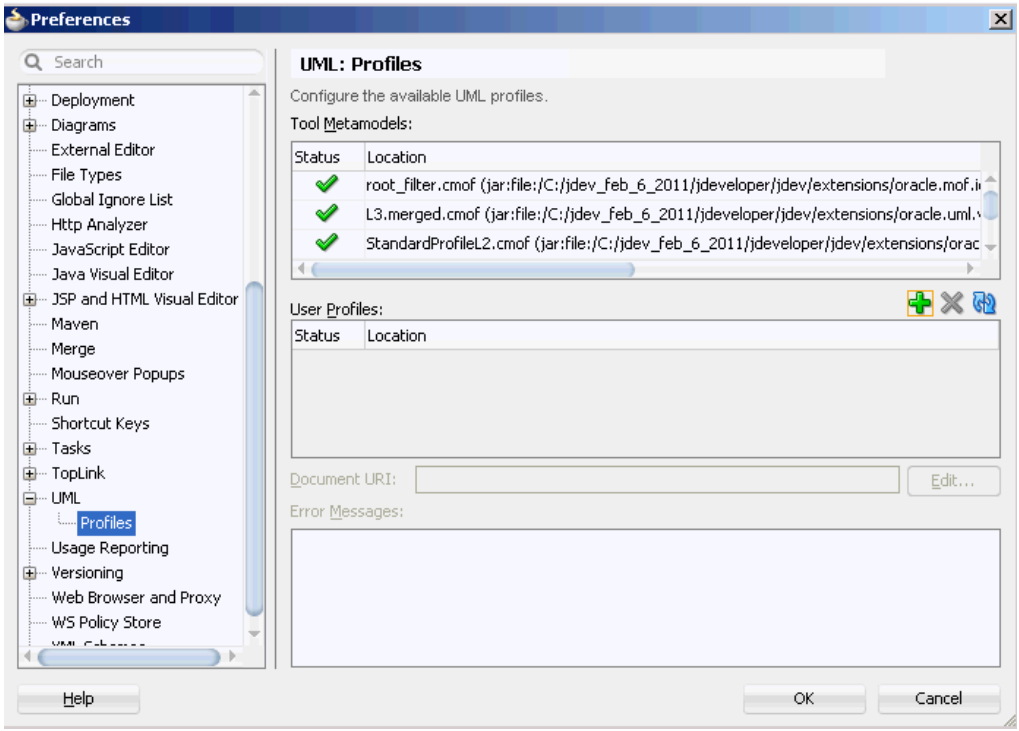
1. Select the project containing the profile and select **File > Export**.
2. Choose Export UML as XMI.
3. In the Export dialog, choose a location and select MOF 2.4.1 XMI 2.4.1 from the version dropdown.

How to Add a Profile

Use Tool Metamodels on the Profiles page for UML to add a UML profile. Select and edit a profile to specify the document URI for the profile.

To add a profile, go to **Tools > Preferences > UML > Profiles**. The profiles page shows all of the current profiles (authored by you or a third party) available in the application, as shown on Figure 5-12. Under Tool Metamodels, click Add (+). Once you have added a profile, select it and click **Edit**. Specify the document URI for the profile. The document URI is used when persisting references to elements in the profile, in particular a profile application's reference to the profile.

Figure 5-12 UML Preferences Profiles Dialog

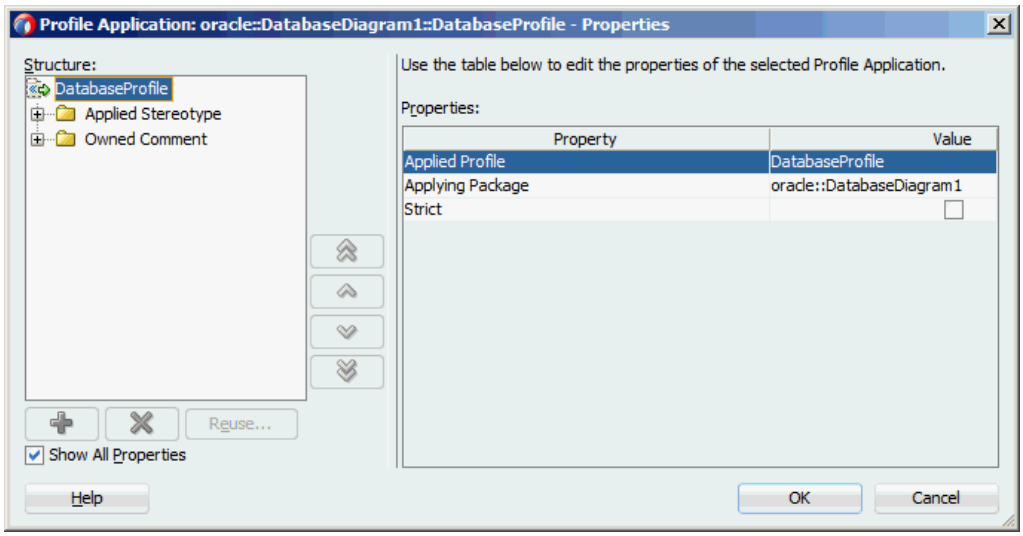


How to Apply a Profile to a UML Package

Use the procedure to apply a profile to a UML package.

Follow these steps to apply a profile to a UML package.

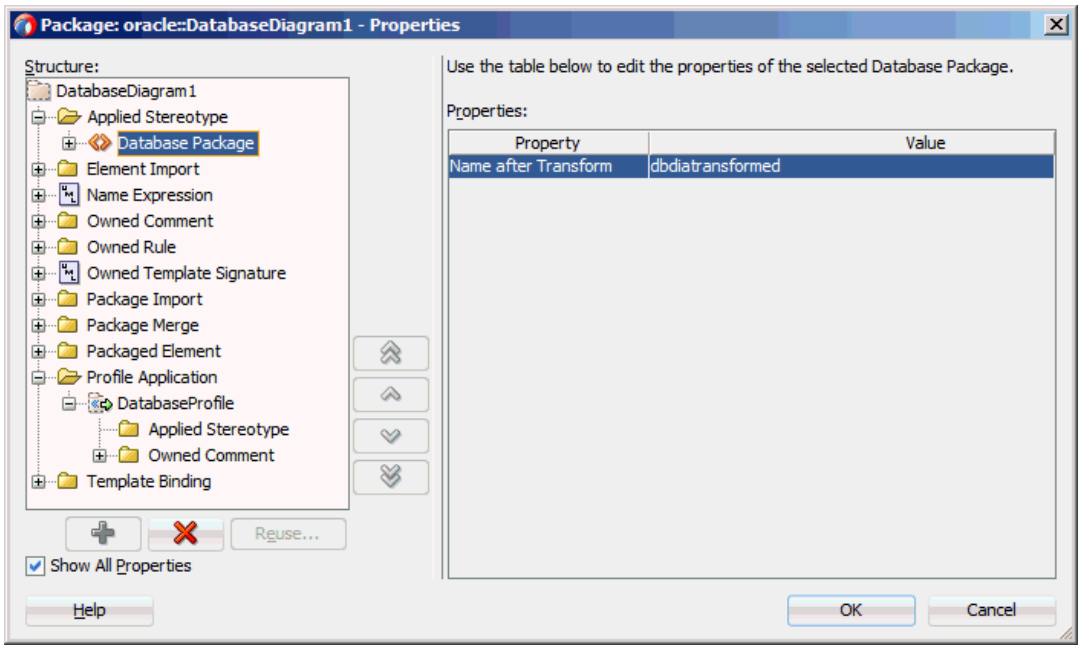
1. In the Applications window right-click on the UML package and choose **Properties**.
2. In the Package Properties dialog, Structure list select Profile Application. Click **Add** (+) and choose a UML profile. Edit any property values and click **OK**.
3. In the Structure list, select **Packaged Element**.



4. Expand a metaclass instance, choose **Applied Stereotype** and click **Add** (+) and select a property. The properties available depend on the UML profile you are using.

   For example, expand the Class node, select **Applied Stereotype**, and click **Add** (+). Figure 5-13 shows the Name after Transform property from the UML profile DatabaseProfile.

Figure 5-13 Package Properties Dialog Showing Resulting Class Name for UML Transformation



Importing and Exporting UML

UML models that are compliant with UML versions 2.1.1 to 2.4.1 can be imported into JDeveloper using XML Metadata Interchange.

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 to 2.4.1 compliant.

The XMI specification describes how to use the metamodel to transform UML models as XML documents.

How to Import and Export UML Models Using XMI

If profiles are in separate files, you can use XMI and import one or more files other than diagrams into JDeveloper. Use the procedure to import a UML model.

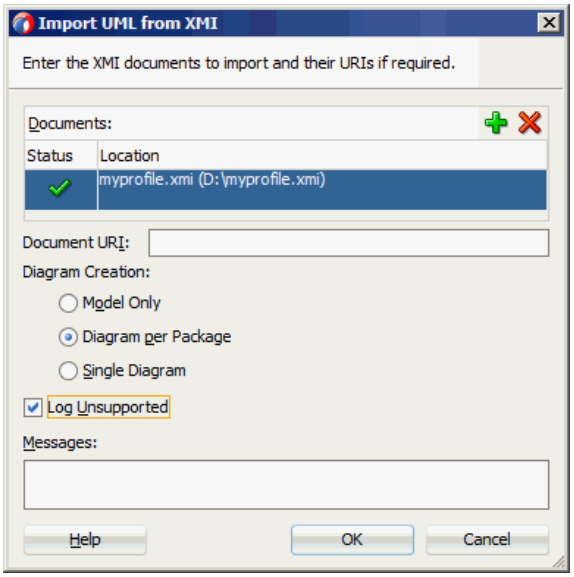
The following are restrictions that apply to importing:

- Diagrams cannot be imported.
- You can use XMI to import one or more files. Any profiles referenced by XMI must first be registered with JDeveloper using **Tools > Preferences > UML > Profiles**.
- Each profile must be in a separate file. For more information see, Using UML Profiles.

To import UML model from XMI:

1. With an empty project selected in the Applications window, choose **File > Import.**
2. Select **UML from XMI**, then click **OK**.
3. Complete the Import UML from XMI dialog. Diagrams can be automatically created from the imported model. The dialog will offer you this option during the process, as shown in Figure 5-14.

Figure 5-14 Import UML from XMI Dialog



How to Export UML Models as XMI

Use the procedure to export a UML model as XMI.

Follow these steps to export a UML model as XMI.

1. Select your project.
2. Choose **File > Export**.

Figure 5-15 Choose UML as XMI Dialog

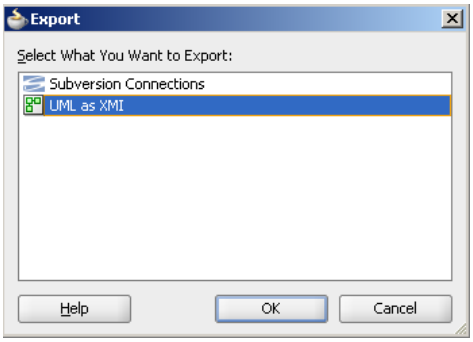
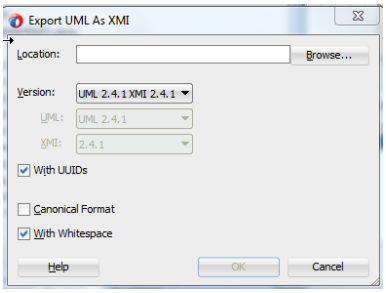


Figure 5-16 Export UML as XMI Dialog



Typical Error Messages When Importing

Incorrect namespaces and standard object references often cause errors and warnings that surface in the XMI Import Log during the importing of UML models. Tabulated are the resolutions to the errors.

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 detailed in Table 5-1. Double clicking on the items in the log navigates to the problem element. Often issues arise because of incorrect namespaces and standard object references.

As with other XML, the structure of a valid file is specified by XML schemas, 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.

Table 5-1 Typical Error Messages When Importing

Error Type	Error Detail	Resolution
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 messages above indicate that a relevant profile has not been registered. To register a profile see, 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`

Table 5-1 (Cont.) Typical Error Messages When Importing

Error Type	Error Detail	Resolution
Invalid L2 Standard 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 L3 Standard 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	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	• `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`

Using MOF Model Libraries

UML objects in a project can be reused by another project by means of Meta-Object Facility Model Library .jar files. Include UML objects in a library using the JDeveloper format, or using an `.xmi` file in which each object has a unique id.

MOF (Meta-Object Facility) Model Library `.jar` files enable UML objects from one project to be reused by another.

UML objects can be included in a library in one of two ways:

- Using JDeveloper's own format
- Using an `.xmi` file supplied by a third party, where each object must have an ID that is unique within the file.

In the latter case you could import the XMI file to convert it into JDeveloper's format. However, this does not preserve object identifiers, which might be necessary if you want to be able to export your own models with standardized references to the library objects.

You might find it useful to reference the model libraries found in the JDeveloper installation:

- `.../oracle/jdeveloper/jdev/extensions/oracle.uml.v2`

  The two ZIP files in this directory use catalogs.

- `.../oracle/jdeveloper/jdev/extensions/oracle.jdeveloper.db.modeler.transform`

  The transform ZIP uses the JDeveloper format.

How to Create an XMI Catalog File

Specify an alias for a library file in the form of a `.xml` Catalog file. Use a text editor to create a `.xmi` file.

When library objects only have IDs, references to them include the URL of their containing file. By default this is the physical location on disk, which might be machine specific. Instead an alias should be specified for the file in the form of an `.xml` Catalog file.

Use text editor to create a `.xmi` file with the following information:

```
<?xml version="1.0" encoding="UTF-8" ?><!DOCTYPE catalog PUBLIC "-//OASIS//DTD XML
Catalogs V1.1//EN"  "http://www.oasis-open.org/committees/entity/release/1.1/
catalog.dtd"><catalog xmlns="urn:oasis:names:tc:entity:xmlns:xml:catalog">  <uri
name="http://example.oracle.com/MyLibrary.xmi" uri="MyLibrary.xmi"/>  <uri name="http://
example.oracle.com/MyLibraryAlias.xmi" uri="http://example.oracle.com/MyLibrary.xmi"/></
catalog>
```

The first URI element maps a standard name to the library file, which is specified relative to the location of the catalog in the library. There should be one of these entries for each `.xmi` file. Optionally, further URI elements such as the second element above, can be used to specify additional aliases for the standard name. This can be useful if there are multiple versions of the library.

Note that only the above URI element from the OASIS XML Catalog specification is supported.

Create a MOF Model JAR File

Use the procedure to create an MOF model JAR file.

To create a MOF model JAR file:

1. Right-click the project that contains the model and choose Properties. In the Project Properties dialog, choose Libraries and Classpath.

   Click **Add Library** or **Add JAR/Directory** to add any `.xmi` and catalog `.xml` files to your project's model path. Click **OK**.

2. Choose **File > New > From Gallery**. From the General category, select Deployment Profiles. Choose the MOF Model Library item and click **OK**.

   a. Specify the name of your deployment profile and click **OK**. The MOF Model Library dialog opens with JAR Options selected on the left.

   b. In the JAR options area, browse to the location of your JAR file and complete the dialog. Click **OK**. You return to the Project Properties Deployment page.

3. In the Deployment Profiles area, double-click the name of your MOF Model Library. Select Library Options in the panel on the left.

   - Enter a name in the Library Name field. It does not have to be the same as the deployment profile name.
   - If you have a catalog file, fill in the MOF Catalog Entry field to specify the catalog location within the JAR file. You can check the file groups to see what will be included in the JAR file.
   - Click **OK**.

4. To deploy your changes to the JAR you have just created, right-click the project and choose **Deploy > MyMOFLibrary**.

Add a MOF Model Library

Adding a library to a project requires the addition of the source path for the MOF Model Library. Use the procedure to add an MOF model library.

To add a MOF model library

If you want to add a library to your project add the source path for your MOF Model Library as detailed here. If you redeploy, you can update the JAR with any new changes you make.

1. Select your project and choose **Tools >Manage Libraries**. Select the Libraries tab and click the **New...** button.
2. In the Library Name field supply your MOF model library name.
3. Add the root model folders of the JAR to the source path. Right-click the project and choose Properties. Select the Project Source Paths node and in the Java Source Paths area on the right, click the **Add** button and browse to the library path, and click OK.

Use a MOF Model Library

Use the procedure to use an MOF model library.

To use a MOF model library

1. Create a new project or select an existing project.
2. View the project properties and choose the Libraries and Classpath node.
3. Click **Add Library** and browse to the location of your MOF model library.
4. Click **OK**.

Using Transformations

The UML transformations on database tables, Java classes, and interfaces that JDeveloper supports can be performed multiple times using the same source and target models. Use the procedure to perform a transformation.

JDeveloper supports UML transformations on your database tables, Java classes, and interfaces. See Transformation Types.

You can perform transformations multiple times using the same source and target models. There are also some reverse transformations to UML from Java classes.

To perform a transformation:

- Select UML objects or elements, database tables, or a Java class.
- From the main menu select **Diagram > Transform** (the Diagram menu is only visible when a suitable element is selected).
- Select a transformation type.

Transformation Types

JDeveloper supports Model Only, Same Diagram, and New Diagram transformation types. Upon a transformation start, select a transformation type.

Once you start the transformation you must choose one of three transformation types:

- – **Model Only**. Creates the definitions of the transformed elements, but does not add them to a diagram. The model is created, then displayed in the Applications window in the current project.
  – **Same Diagram**. Creates transformed elements in the current diagram. The created model can be viewed in:
    * The Applications window.
    * The UML Class diagram. If the conversion is from a database, it will be visible as modeled tables and constraints.
  – **New Diagram**. Creates a new diagram for the transformed elements. If this is a database, the new class diagram will have classes for each transformed database table.

UML-Java Transformation

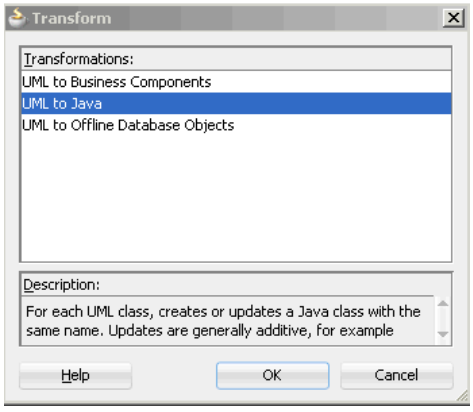
You can use the UML modeling tools to create a UML Class model, and then transform it to Java, or vice-versa.

Transform UML to Java

Use UML modeling tools to create a UML Class model, and then transform it to Java. Use the procedure to transform UML to Java.

1. Select the UML elements to transform.
2. Select the transformation type.
3. Click **OK**.

Figure 5-17 Transform Dialog Showing UML Options

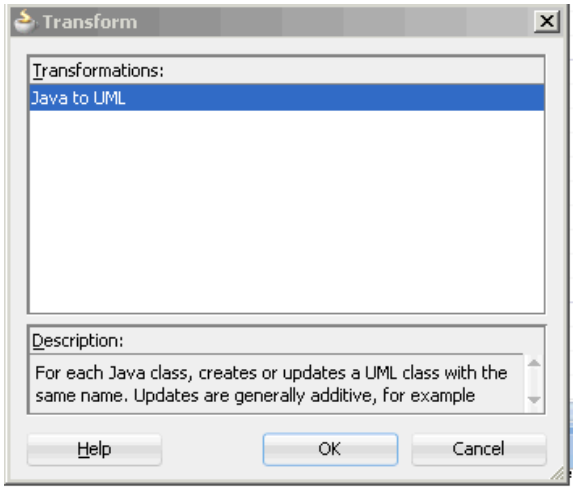


Transform Java to UML

UML modeling tools are used to create a UML Class model, which is then transformed to Java, and vice versa. Use the procedure to transform Java to UML.

1. Select the Java Element to transform and select Transform from the context menu.
2. Select the transformation type.
3. Click **OK**, as show in Figure 5-18.

Figure 5-18 Transform Dialog Showing Java to UML Options



UML-Offline Database Transformation

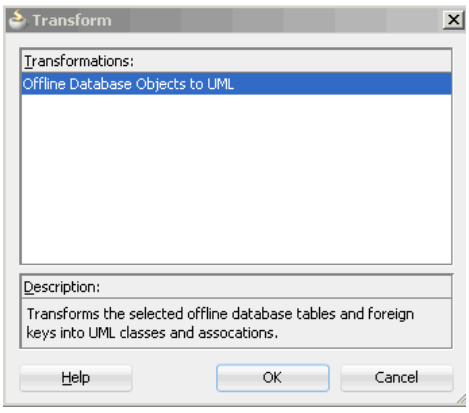
You can use the UML modeling tools to create a UML Class model, and then transform it to Java, or an offline database.

Transform an Offline Database Diagram to UML

UML modeling tools can be used to create a UML Class model, which can then be transformed to Java, or an offline database. Use the procedure to transform an Offline Database diagram to UML.

1. Select the offline database object or objects you want to transform.
2. Right-click and choose **Transform**.
3. Select the transformation type.
4. The Transform dialog appears as shown in Figure 5-19. Select **Offline Database Objects to UML**.
5. Click **OK.**

Figure 5-19 Transformation Dialog Showing Offline Database to UML Dialog Option



Transform UML to Offline Database Objects

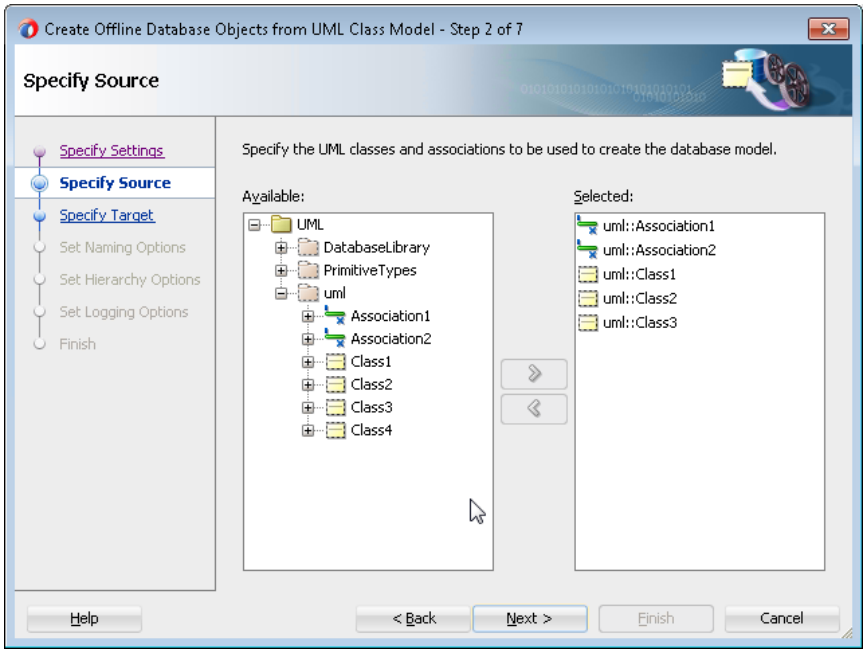
UML modeling tools can be used to create a UML Class model, which can then be transformed to Java, or an offline database. Use the procedure to transform UML to an offline database object.

1. In the Applications window, select the project containing the UML classes to transform.
2. Choose **File > New > From Gallery**.
3. Expand the **Database Tier** and choose Offline Database Objects.
4. From the items list, choose **Offline Database Objects from UML Class Model** and click **OK**.

   The Offline Database Objects from UML Class Model wizard opens.

   When you invoke the wizard from the New Gallery, the offline database objects created during the transformation process are only available in the Applications window.
5. **Specify settings**.
   - **Default**. Resets the settings in Table 5-2 to default values.
   - **Previous Transformation**. When you have previously run the wizard in this session of JDeveloper you can select this option to use the last set of transform settings.
   - **Previously saved file**. Select to use transform settings saved to a file.
6. **Specify source**. Expand the folders on the left, select the elements to be transformed for the database model and use the > arrow to move them to the Selected list. See Figure 5-20.

Figure 5-20 Create Offline Database from UML Class Model Example



7. **Specify Target**. Specify the target database and its schema. Click **Help** for an explanation of each option.
8. **Set Naming Options**. Select the UML name conversion formats.

9. **Set Hierarchy Options**. Specify the transformation rule and enable/disable creation of an intersection table. Click **Help** for an explanation of each option.
10. **Set Logging Options**. Select the messages to be logged.
11. **Inspect a preview**. The final page of the wizard gives you the opportunity to view the database model, and to save the transformation settings to a file.
12. Click **Finish**. The offline database objects are created and displayed in the Applications window. The transformation settings are saved (if you have enabled that option).

Transform UML Classes on a Diagram to an Offline Database

Use the procedure to transform a UML class to an offline database object.

1. Open the diagram to transform.
2. Select the elements to transform. Right-click and choose **Transform.**
3. Select the transformation type.
4. Choose **UML to Offline Database.**
5. The transform wizard opens and you can select the options for the transformation.

Set Hierarchy Options

The Set Hierarchy Options page on the Create Offline Database Objects from UML Class Model wizard provides options to transform root and leaf classes to offline tables. Transform all classes with inheritance and Transform all classes, creating foreign keys create an offline table for every class in the transform set.

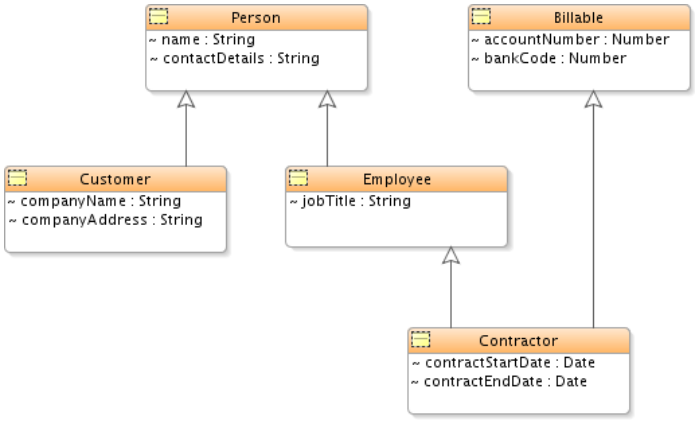
There are four types of generalization to specify on the Set Hierarchy Options page on the Create Offline Database Objects from UML Class Model wizard.

The options are:

- Transform root classes
- Transform leaf classes
- Transform all classes, with generalization
- Transform all classes, creating foreign keys

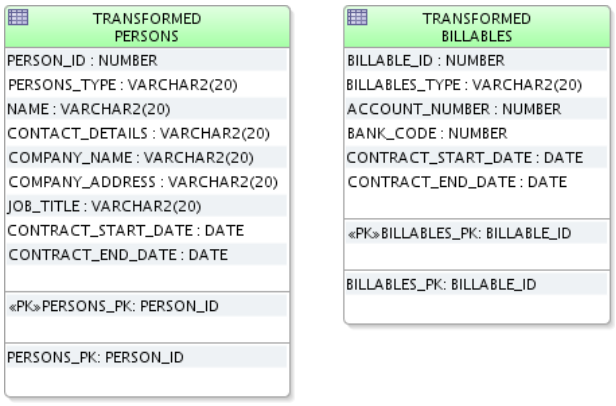
Consider the case of two root classes and three leaf classes, as shown in Figure 5-21.

Figure 5-21 Diagram Showing Two Root and Three Leaf Classes



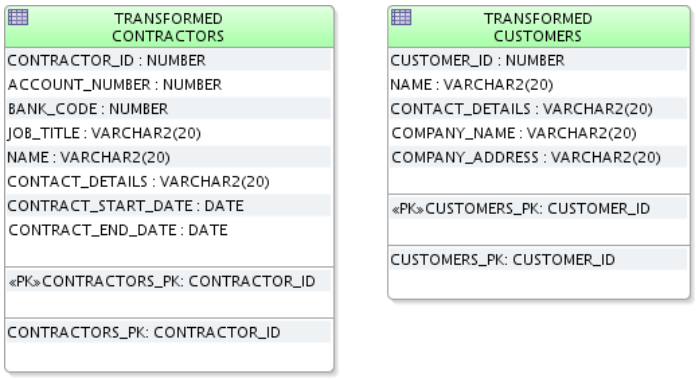
If you select the option **Transform Root Classes**, root classes are transformed into offline tables, and all the attributes and foreign keys from their descendant classes in the hierarchy are also transformed as shown in Figure 5-22. You also have an option of creating a discriminator column. The discriminator column contains marker values for the persistence layer to decipher what subclass to instantiate for a particular row.

Figure 5-22 Transformed Root Classes



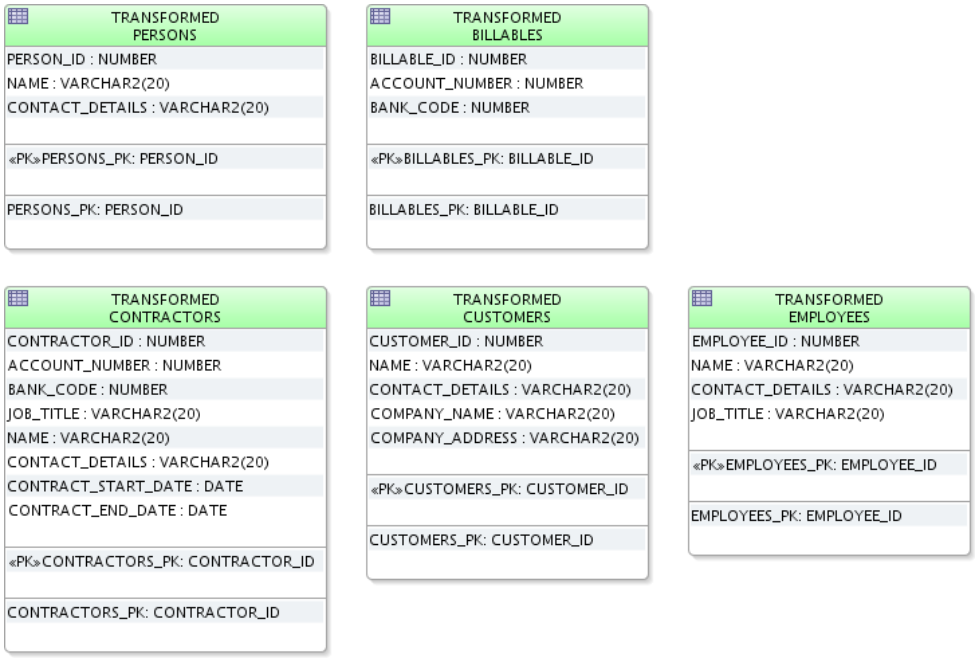
If you select the option **Transform leaf classes**, leaf classes are transformed into offline tables, and they inherit their columns and foreign keys from their ancestor classes, as shown in Figure 5-23.

Figure 5-23 Leaf Classes Transformed Inheriting From Generalized Classes



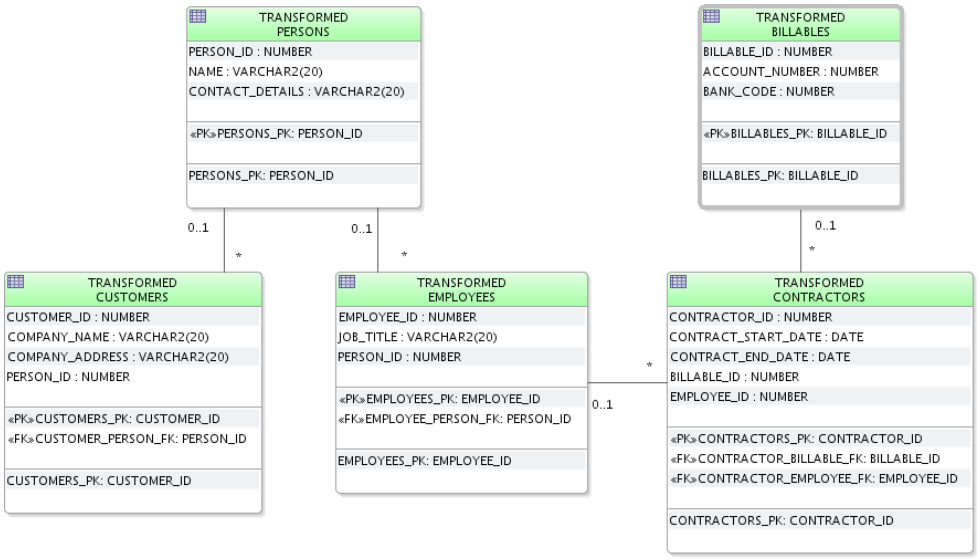
If you select the option **Transform all classes, with inheritance**, an offline table is created for every class in the transform set. Each table inherits columns and foreign keys from its ancestor tables but is otherwise independent, as shown in example Figure 5-24.

Figure 5-24 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 5-25.

Figure 5-25 All Classes Transformed with Foreign Keys



Transformation Settings

Tabulated are the transformation settings that can be saved when the Save Transform Settings to File option is selected on the final page of the Create Offline Database Using Transformations Objects from UML Class Model wizard.

Table 5-2 shows the information that can be saved when you choose the Save Transform Settings to File option on the final page of the Create Offline Database Objects from UML Class Model wizard. See Step 12 in Transform UML to Offline Database Objects.

Table 5-2 Transform Settings That Can Be Saved

Transform Settings	Page in Wizard and Notes
Target project and offline database	Specify Target. Specify where to transform the UML objects to.
Offline schema	Specify Target. The schema to transform the objects into. Note that if you select Choose Schema and specify a schema which does not exist it will be created. If you transform once to a new schema, rename the schema then transform again reusing the same settings the old schema is recreated and the transformed objects will be created there.
UML name conversions	Set Naming Options. Specify the rule for capitalizing and pluralizing UML names.
Invert role names	Set Naming Options. Specify whether to use the role names at the adjacent or opposite ends of a UML association when creating foreign key columns.
Add comment option	Set Naming Options. Specifies whether the new offline database objects should have comments explaining how they were created.
Hierarchy options	Set Hierarchy Options. Determines how classes in a hierarchy should inherit attributes and associations when they are transformed into tables.
Many-to-many associations	Set Hierarchy Options. Specifies whether to create intersection tables for many-to-many associations.
Log messages	Set Logging Options. Specifies type of messages to log (Error, Warning, Information, Create and Progress).

Reuse Transform Settings

Follow the steps to reuse transformation settings using the Reapply Settings option.

You can specify the transformation settings as discussed in step 5 of Transform UML to Offline Database Objects.

1. In the Applications window, select a class and from the context menu, select **New > Database Objects from UML Class Model**.
2. On the Specify Settings page, specify the transformation settings. If you make a change on the Settings page, and continue using the wizard, you can use the Back button to return to the Specify Settings page. In this case you will see the **Reapply Settings** option, which you can choose to change the initial settings for this wizard.

Using DatabaseProfile for UML Transformations

The UML profile called `DatabaseProfile` determines how class models are transformed to offline database models. Tabulated are the stereotype properties of `DatabaseProfile` that control how elements are transformed.

JDeveloper comes with a UML profile called DatabaseProfile which determines how class models are transformed to offline database models. See Using UML Profiles.

DatabaseProfile contains stereotype properties that control how elements are transformed. The stereotypes and their properties in this profile are described in Table 5-3.

Table 5-3 Stereotypes and Properties in DatabaseProfile

Stereotype	Applied to	Offline Database Type	Properties	Notes
Database Package	UML::Package	SCHEMA	Name after transform	
Database Class	UML::Class	TABLE	Name after transform	
Database Attribute	UML::Property	COLUMN	Datatypes, Primary	A UML Property can be an attribute or an association end.
Database Datatype	UML::Type	n/a	Datatype	In the 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.
Database Constraint	UML::Constraint	CONSTRAINT	Name after transform, Body	Creates a check constraint against the transformed table.
Database Association	UML:: Association	CONSTRAINT	Foreign Key naming rule	
Database Generalization	UML:: Generalization	CONSTRAINT	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 5-4.

Table 5-4 Properties of Stereotypes in DatabaseProfile

Property	Description	Type
Name after transform	The name of the transformed database object. If blank, default naming rules are applied.	String
Body	The SQL code for the check constraint.	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 derive a default name from the table names.	Both Tables Database Name Owning Name UML Name

Table 5-4 (Cont.) Properties of Stereotypes in DatabaseProfile

Property	Description	Type
Primary	Flag to indicate that transformed column is part of the primary key for the parent table.	Boolean

Use DatabaseProfile to Transform a Class Model

Use the procedure to transform a class model using `DatabaseProfile`.

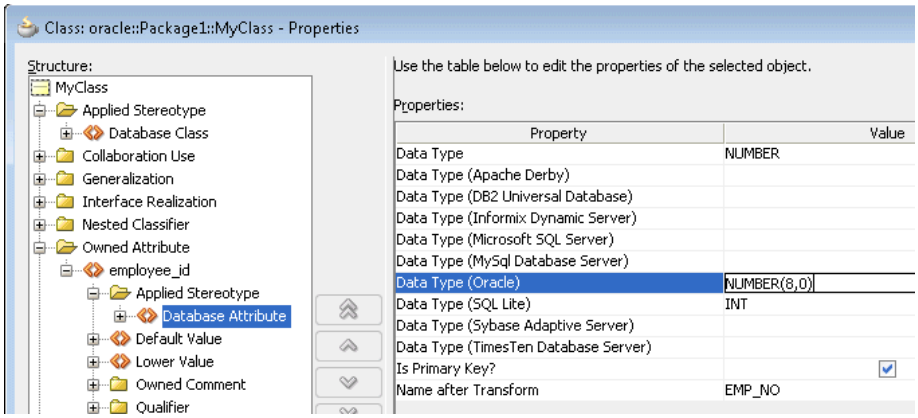
You can transform a class model using DatabaseProfile.

1. In the application window, right-click on a UML package (`package.uml_pck`) and choose **Properties**.
2. In the Package properties Structure pane, select **Profile Application** and click **Add** (+).
3. Select `DatabaseProfile` from the list of available profiles. Click **OK**. A new file `DatabaseProfile.uml_pa` is now listed in the Applications window.
4. Now you can apply stereotypes to the various elements in the project. 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 5-13.
5. Once you have set the stereotypes to apply, proceed to transform the UML Class model following the steps in UML-Offline Database Transformation. The stereotypes and properties in DatabaseProfile you set are applied during transformation. See Table 5-2.

You can apply stereotypes to other elements as well. For example, you can specify datatypes and a primary key to attributes owned by a particular class. In the same Class Properties dialog, expand **Owned Attribute**. 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 5-26 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 5-26 Database Attributes for Applied Stereotypes



For information about the stereotypes and properties covered by DatabaseProfile, see Table 5-3 and Table 5-4.

Logging Options

Follow the steps to select the actions that you want logged.

1. Select a class diagram and from the context menu, select New > Offline Database Objects from UML Class Model. The wizard opens.
2. On the Set Logging Options page, choose the type of actions you want logged.

UML-ADF Business Components Transformation

Use the procedure to transform a UML class diagram to an ADF Business Component.

If you have a UML class diagram it can be transformed to ADF Business Components using much the same process described in UML-Offline Database Transformation.

1. In the Applications window, select the project containing the UML classes to transform.
2. Right click on the class to be transformed and select **Transform** from the context menu.
3. Select the transformation type.
4. You are prompted to create a database to contain the transformed components. After the database connection the creation of the ADF Business component proceeds in the same manner as the creation of a new ADF Business component diagram.

Modeling with UML Class Diagrams

In UML class diagrams, rectangles represent classes that along with interfaces contain compartments that hold only one type of information. The table provide descriptions of elements in class diagrams.

In UML class diagrams classes are represented as rectangles containing class name and details. On the diagram, classes and interfaces are divided into compartments, with each compartment containing only one type of information, as shown in Figure 5-27. The possible elements are listed in Table 5-5. Classes can be displayed as compact, symbolic or expanded nodes.

Figure 5-27 UML Class Diagram with Elements

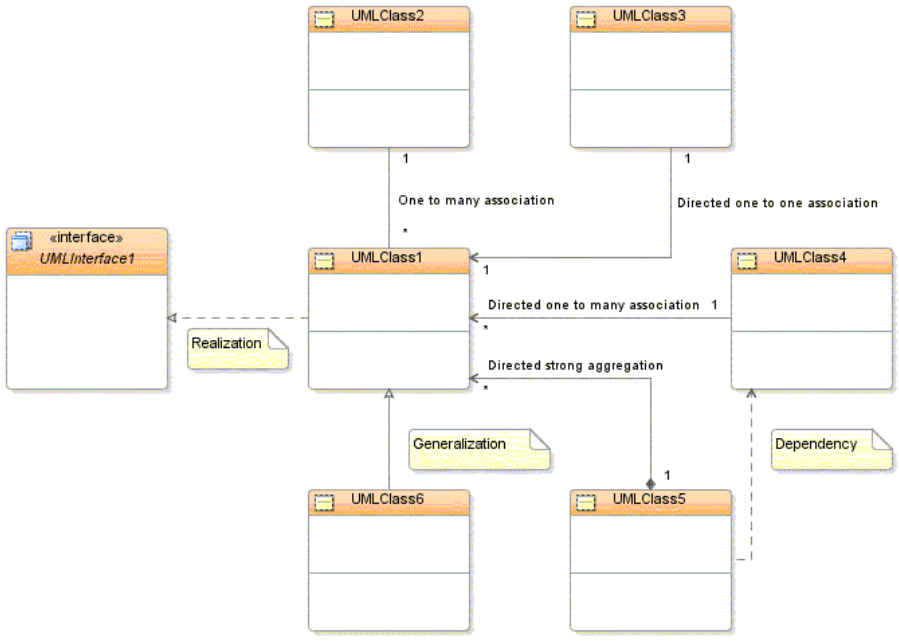


Table 5-5 Elements in Class Diagram

Elements	Description
* to * Association	Defines a many-to-many relationship between UML classes.
1 to * Association	Defines a one-to-many relationship between UML classes.
1 to * Association Class	Defines a non-directed one-to-many association class.
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.
Constraint	Constraints are the degree of freedom, or lack thereof that you have in modeling a system behavior, or solution.
Constraint Attachment	Attaches constraints to other UML elements.
Data Type	Data types are modeled elements that define your data values.
Directed 1 to * 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 1 to 1 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 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.
Enumeration	Enumerations are data types with a finite, and normally small, set of named literals. Enumerations contain sets of named identifiers that are its values.
Generalization	Defines generalization relationships between classifiers. Represented on the diagram as a solid line with an empty arrowhead pointing towards the specialized classifier or interface.

Table 5-5 (Cont.) Elements in Class Diagram

Elements	Description
Interface	Interfaces are represented with a keyword in the name compartment: «interface». Interfaces can be displayed as compact, symbolic, or expanded nodes. Nested classes and interfaces can be modeled inside standard and expanded interfaces.
Package	Use to divide a system into multiple packages, which can simplify and make the system easier to understand.
Primitive Type	Primitive types or data types are data types such as boolean, byte, decimal, DateTime, Double Float, and Time.
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.

Creating a UML Class Diagram

Use the procedure to create a UML class diagram. Set properties for diagram display using the Diagram option for Class under Diagrams for tool preferences.

Follow these steps to create a UML class diagram:

1. Right-click the project and select **New > From Gallery**.
2. In the General category, select the UML node. In the Items pane, select Class Diagram, and click **OK**.
   a. Give the diagram a name that is unique within this package.
   b. Select or browse to the owning package for this diagram.
3. Click **OK**.

Figure 5-28 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.

To set the diagram display properties go to **Tools > Preferences > Diagrams > Class** and choose **Diagram**.

You can also change the way elements are represented on a diagram. 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**.

Working with the Class Diagram Features

Drag a class onto a diagram to create classes and interfaces. Class properties can be added to modeled classes and interfaces on a diagram.

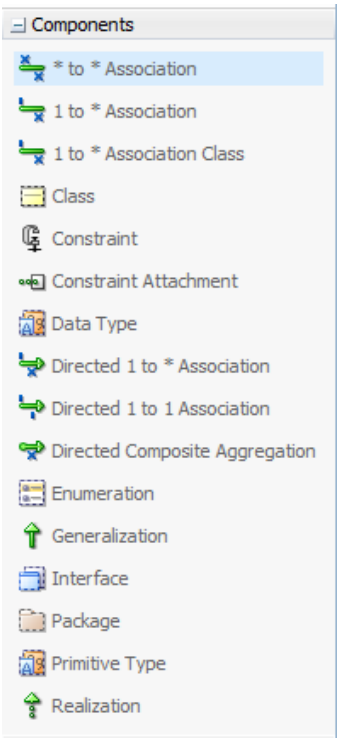
The components window contains the elements available for your class diagram, as shown in Figure 5-28 and described in Table 5-5.

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.

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**.

Figure 5-28 Class Diagram Components window



How to Create Classifiers, Constraints, and Packages

Create classifiers, constraints and packages using Class from the Components window. The new object is created in a package that corresponds to the diagram folder.

You can create classifiers, constraints and packages in the same way you create diagram nodes, as discussed in Working with Diagram Nodes. except choose Class from the Components window dropdown. The new object is created in a package that matches the folder in which the diagram is contained.

How to Create Attributes

An attribute can be created from a modeled class or interface by using the element property dialog. Dragging an attribute from one class or interface to another class or interface on the same diagram also creates an attribute.

There are several ways to create an attribute:

- Double-click the modeled class or interface, then add the attribute using the element property dialog.
- Right-click the class or interface and choose **Properties,** then add the attribute using the element property dialog.

- Drag an existing attribute from one class or interface on a diagram to another class or interface on the same diagram.

To arbitrarily change the order of an attribute within a class, disable the Sort Alphabetically option and drag the attribute up or down on the screen. Select **Tools > Preferences**, open the **Diagrams** node and choose **Class**. From the **Edit Preferences for:** dropdown, select Class or Interface. Click the **Attributes** or **Operations** tab and deselect **Sort Alphabetically**).

How to Add Nested Classes and Nested Interfaces

Nested classes and interfaces are created in the modeled class or interface by using the Expanded option from View As in the context menu of the class.

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.

How to Add Attributes and Operations

An attribute or operation can be added from a modeled class or interface by using the element property dialog. Dragging an attribute or operation from one class or interface to another class or interface on the same diagram also adds an attribute or operation.

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**).

How to Hide Attributes and Operations

Use the procedure to hide one or more attributes or operations using the context menu of the diagram elements.

Use the right-click context menu to hide or show attributes and operations elements on your diagram.

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 attributes or operations choose **Show All Hidden Members**.

How to Add Generalizations, Realizations, and Associations

Use Generalization in the Class Components window to create generalized structures on a diagram of classes. Associations are created between modeled classes and interfaces using the association icons.

Generalized structures are created on a diagram of classes by using the Generalization icon on the Class Components window.

Where an interface is realized by a class, model it using the Realization icon on the Class Components window 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.

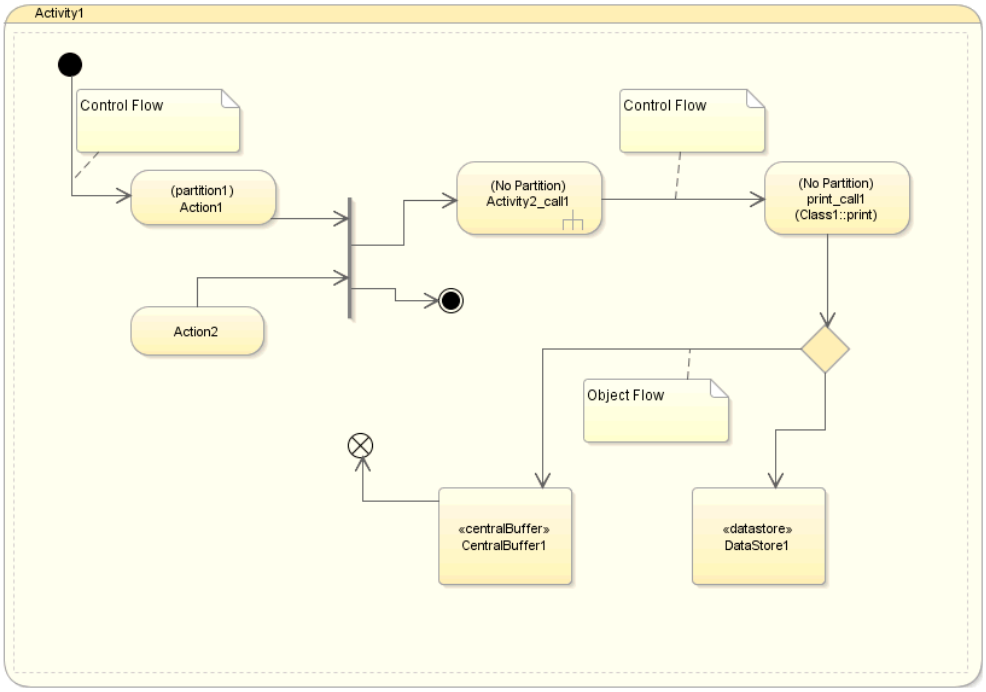
Modeling with Activity Diagrams

Activity diagrams capture the behavior of a system. Model business processes with activity diagrams by showing the coordinated execution of actions.

Use activity diagrams to model your business processes. Your business processes 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 the following figure.

Figure 5-29 Sample Activity Diagram Showing Elements



Working with the Activity Diagram Features

Elements available for an activity diagram are placed within the Components window. Activity can be placed directly on the diagram, and other elements can be placed inside an Activity.

The Components window contains the elements available for 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 5-30 and Table 5-6.

Figure 5-30 Components window for Activity Diagram

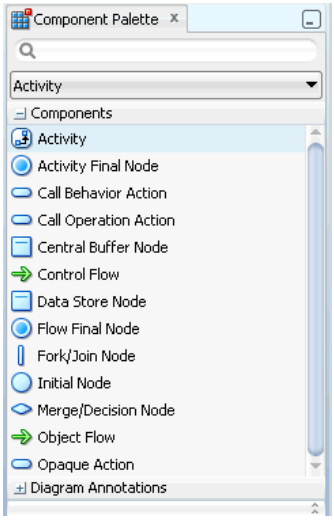


Table 5-6 Activity Diagram Elements

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.

Table 5-6 (Cont.) Activity Diagram Elements

Element	Description
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.
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

How to Create an Activity Diagram

Use the New Gallery wizard to create your activity diagram following the steps in Creating a New Diagram.

How to Create Initial and Final Nodes

Create nodes using the Initial Node, the Activity Final Node, or Final Flow Node options in the Component window.

To create nodes, click the **Initial Node** icon, the **Activity Final Node** icon, or **Final Flow Node** icon in the Component window, then click on the diagram where you want to place the node.

How to Show Partitions

Create partitions on a diagram using Show Activity Partition under Display Options in the Properties window. Use the procedure to show a partition on an activity diagram.

You create partitions on a diagram by selecting an action, then selecting **Show Activity Partition** under Display Options in the Properties window.

To show a partition on an activity diagram:

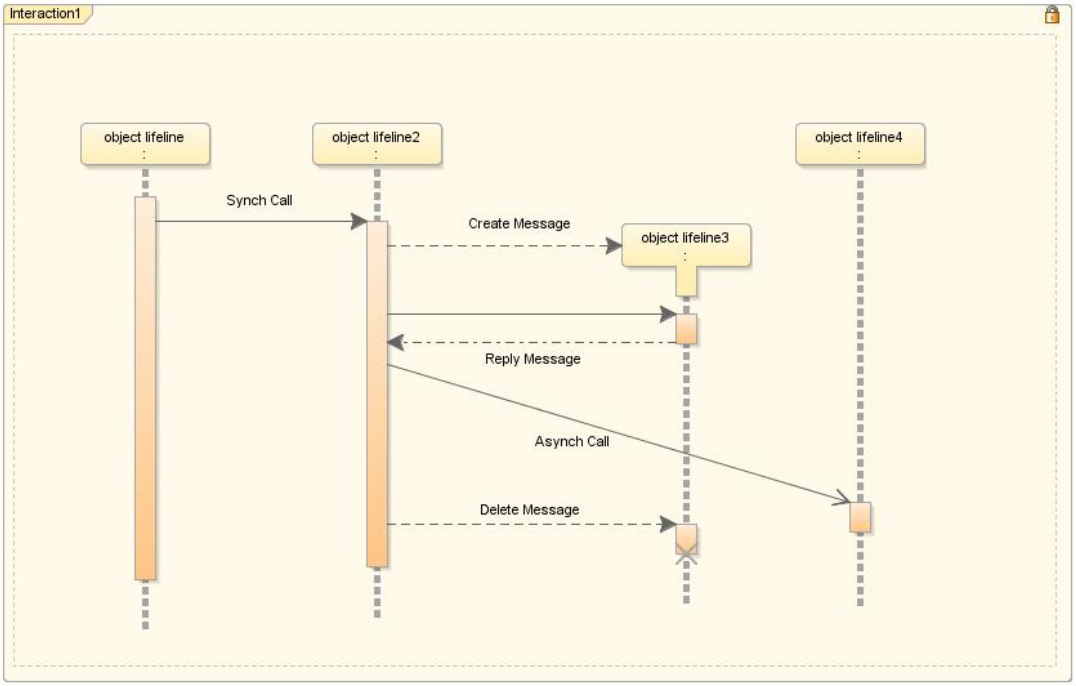
1. In the activity diagram, select an action.
2. In the Properties window, expand the Display Options node.
3. Select **Show Activity Partition**. The action on the diagram displays the text, (No Partition).
4. Click on the text. An editing box appears where you can enter a name for the partition.

Modeling with Sequence Diagrams

Interactions among class instances are modeled as exchanges of messages and represented in 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 the following figure.

Figure 5-31 Typical Sequence Diagram Example



Working with the Sequence Diagram Features

Elements added from the Components window are positioned in a default position on a sequence diagram. Lifelines are aligned vertically, and synchronous and asynchronous calls are chronologically placed on the page.

The elements you add from the Components window are laid out in a default position on your sequence diagram. Lifelines are aligned vertically, unless they are related to another Lifeline, in which case they are aligned with the create message. Synchronous and asynchronous calls and are placed in time order down the page.

The following figure displays the elements in the Components window available to use in your sequence diagram. Each element is represented by a unique icon as well as a descriptive label.

Figure 5-32 Sequence Diagram Components window

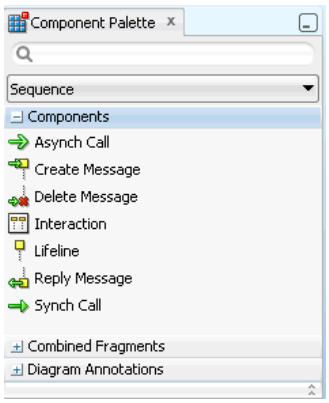


Table 5-7 Sequence Diagram Elements

Element	Description
Asynchronous Call	Represented on a diagram by a diagonal line with an open arrowhead. An asynchronous call 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.
Lifeline	Represented on a diagram by a rectangular box with a vertical dashed line descending beneath it. A 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 calls, 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).

How to Add and Create a Sequence Diagram

Use the New Gallery wizard to create your activity diagram following the steps in Creating a New Diagram.

How to Start a Sequence Tracer

Use the Trace Sequence option from the context menu of a sequence diagram to start a 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.

How to Automatically Layout Elements in an Interaction

You can right-click an Interaction and choose **Sequence**, then **Automatic Layout** to autolayout the elements within the Interaction.

How to Add Lifelines and Classifiers

An interaction, when added to a sequence diagram, facilitates the addition of a lifeline. Use the Attach Classifier option from the context menu of a lifeline to add a classifier.

You add Lifelines to a sequence diagram by first adding an interaction then clicking on the 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 Lifeline and choosing Attach Classifier, which opens a list of elements from which you choose the one you want associated with the Lifeline. Another way to attach a classifier is to drag the classifying object from the databases window onto the Lifeline. These methods are confirmed by the appearance (in the top left of the Lifeline) of an icon representing the classifying element.

How to Create a Synch Call

A synchronous call is added by clicking on Message, the start point for the message, and its destination.

You add a synchronous call 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.

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 call can be the same Lifeline, in which case you have created a self call.

Synch calls are depicted on the sequence diagram by solid lines with filled arrowheads.

How to Work With Execution Specifications

Execution specifications when overlapped on a diagram get merged. You can move an execution specification, and resize an execution specification box.

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.

How to Add a Create Message

Click on Creation Message, the originating object, and the object to create. These actions add a creation message .

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 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.

How to Create a Delete Message

Add a Lifeline for the object that the message deletes. Click Stop or Destroy Message, click on the originating object, and then on the lifeline to delete the message.

Before you add a stop or destroy message, you must already have added an 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 lifeline 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.

How to Create a Reply Message

Click Return, click an end execution specification, and the corresponding start execution specification to add a return message. A return message is editable.

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.

How to Create an Async Call

Click Async Message, the start point for the message, and the destination of the message to add an asynchronous call. The message text is editable.

Add an asynchronous call (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 call can be the same Lifeline, in which case you have created a self call.

Asynchronous calls are depicted on the sequence diagram using diagonal lines and open arrowheads.

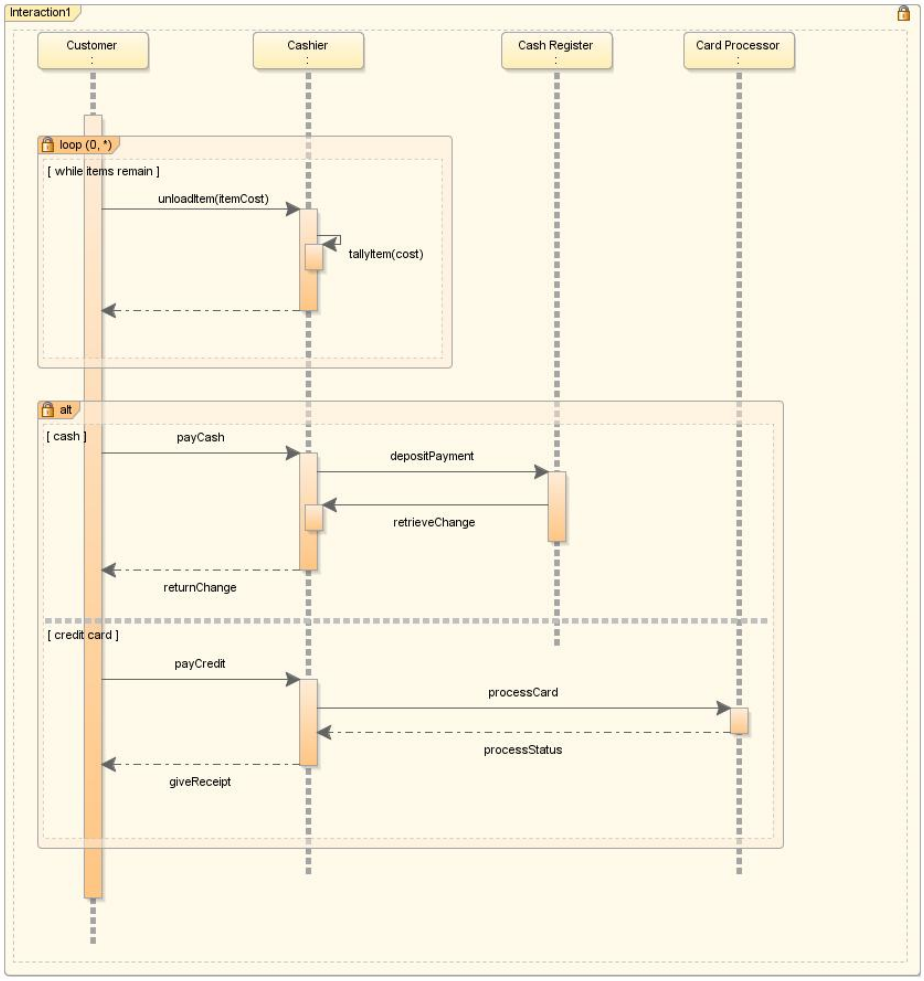
Using Combined Fragments

An interaction operator and corresponding operands define an interaction that is expressed as a combined fragment, and it is displayed in the Components window when the diagram is open. Locked interactions restrict interaction behavior to the interaction on the diagram, and when unlocked, interactions act beyond the interaction and diagram.

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 5-33 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 5-33 Typical Sequence Diagram with Combined Fragments



The following figure shows the combined fragments that display in the Components window when your diagram is open in the diagramming window.

Figure 5-34 Combined Fragments in Components window

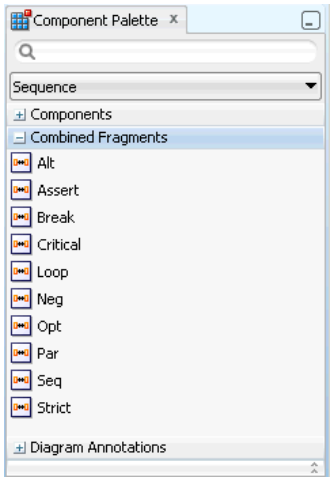


Table 5-8 Combined Fragments Interaction Operators

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.

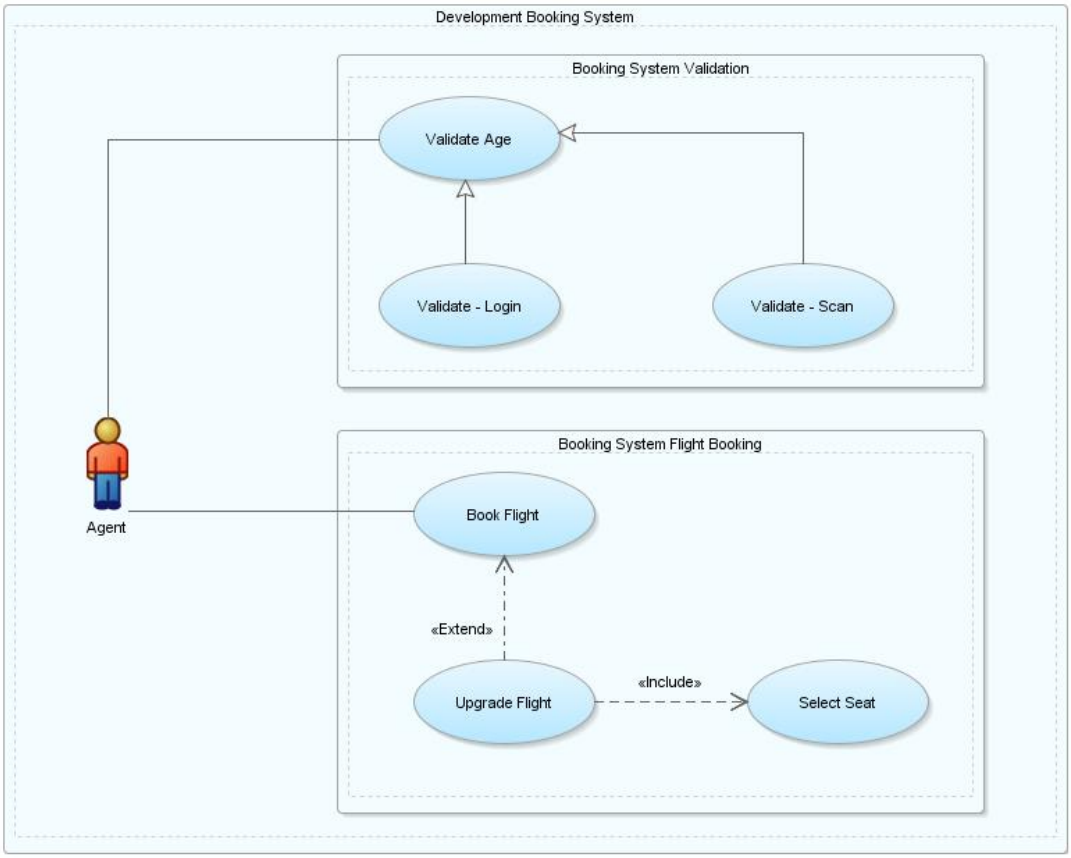
On your sequence diagram interactions you will see combined fragment lock icons. Locking and unlocking an interaction allows you to keep the combined fragment behavior within that interaction on that diagram, or extend its reach to other interactions and other diagrams.

Modeling with Use Case Diagrams

Capture the requirements of a system with use case diagrams.

Use case diagrams capture the requirements of your system, as shown in the following figure.

Figure 5-35 Typical Use Case Diagram



Working with the Use Case Diagram Features

In UML, use case diagrams are behavior diagrams that describe internal and external influences on some system or systems, a set of actions or use cases, and their relationships. JDeveloper displays use case diagrams in the Components window along with available elements.

Use case diagrams express the declared behaviors of your system and how systems and entities interact with it according to subject and actor use cases.

The following figure displays the Components window with the elements available to add to your use case diagram. Each element is represented by a unique icon and descriptive label.

Figure 5-36 Use Case Elements in the Components window

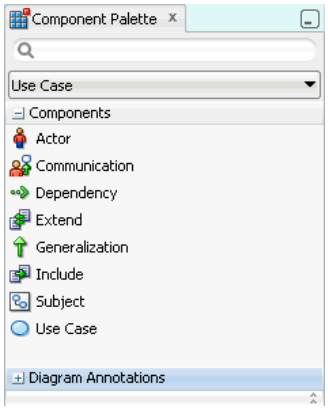


Table 5-9 Use Case Elements

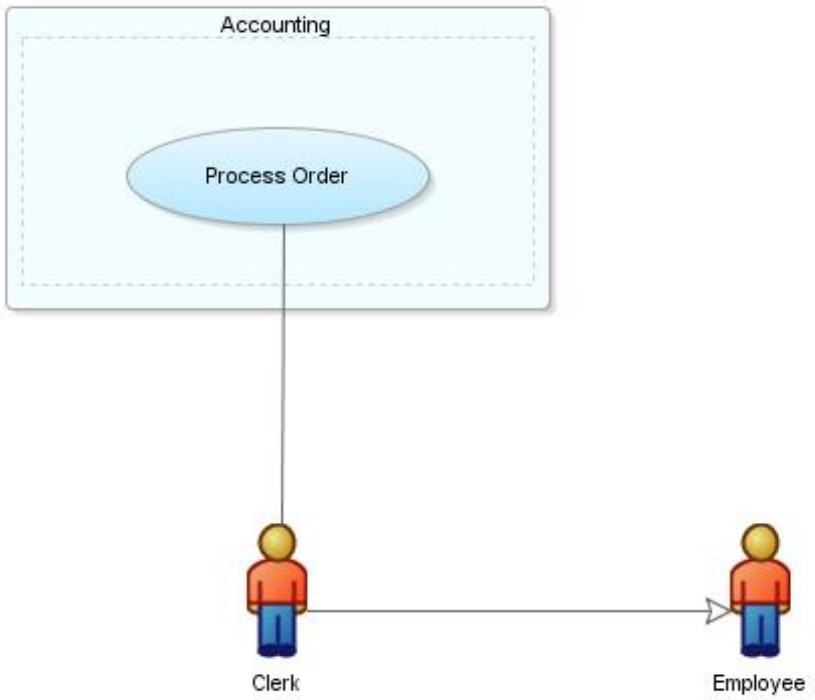
Component	Description
Actor	Represents an abstract role within a system.
Communication	Identifies where an actor is associated with a particular use case.
Dependency	Shows a relationship between one element and another.
Extend	Shows a target use case extends the definition of a source use case.
Generalization	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.

Getting A Closer Look at the Use Case Diagram Elements

Use the Properties window to modify the attributes of elements, such as appearance, in use case diagrams. You can also modify the properties using the context menu of the elements.

You can determine the appearance and other attributes for subject, actor and other objects of these types by modifying the properties in the Properties window, or by right-clicking the object and modifying the properties.

Figure 5-37 Use Case Subject, Actor and object Example



How to Add a Subject to a Use Case Diagram

Enclose all actors and use cases within a subject to depict a system that is being modeled, and enclose groups of use cases within subject lines to indicate development. Use Subject in the Components window to add a subject, and indicate area of influence of the subject in the diagram.

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 Components window, then drag the pointer to cover the area that you want the subject to occupy. 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.

How to Create Actors and Use Cases

Use Actor in the Components window to create an actor on a diagram, and click to indicate the location of the actor. The properties of an actor or use case are editable.

Create actors on a diagram by clicking on the **Actor** icon on the Components window, 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.

How to Represent Interactions Between Actors and Use Cases

The only element that you can add directly to a diagram is Interaction, which encompasses all elements. In the Components window, Communication represents actor-use case interactions, Generalization helps create generalization structures between them, Include shows the inclusion of one use case in another, and Extension represents extensions of use cases.

An Interaction is the only element you can add directly to the diagram. You put all of the other elements within an Interaction.

You can represent interactions between actors and use cases on a diagram using the **Communication** icon on the Components window. 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 Components window under **Diagram Annotations**.

How to Represent Relationships Between Use Cases and Subjects

Communication in the Components window represents interactions between use cases and subjects.

You can represent interactions between use cases and subjects using the **Communication** icon on the Components window.

Exporting a Use Case Model for the First Time

The Use Case modeler generates a set of HTML files from a Use Case model, and the Export to HTML option helps generate HTML files from an existing use case model. View the changes in the model that are to be exported in the Exporting Model dialog.

The Use Case modeler can generate a set of HTML files from a Use Case model. Each HTML file corresponds to one of the Use Case elements. The HTML files are generated in the same project as their relative UML elements, therefore, they are visible in the Application Navigator and they can be opened and edited with an editor.

To generate HTML files from an existing use case model, right-click on the diagram surface and choose **Export to HTML** .

The Exporting Model dialog displays the changes in the model that are about to be exported, as shown in the following figure.

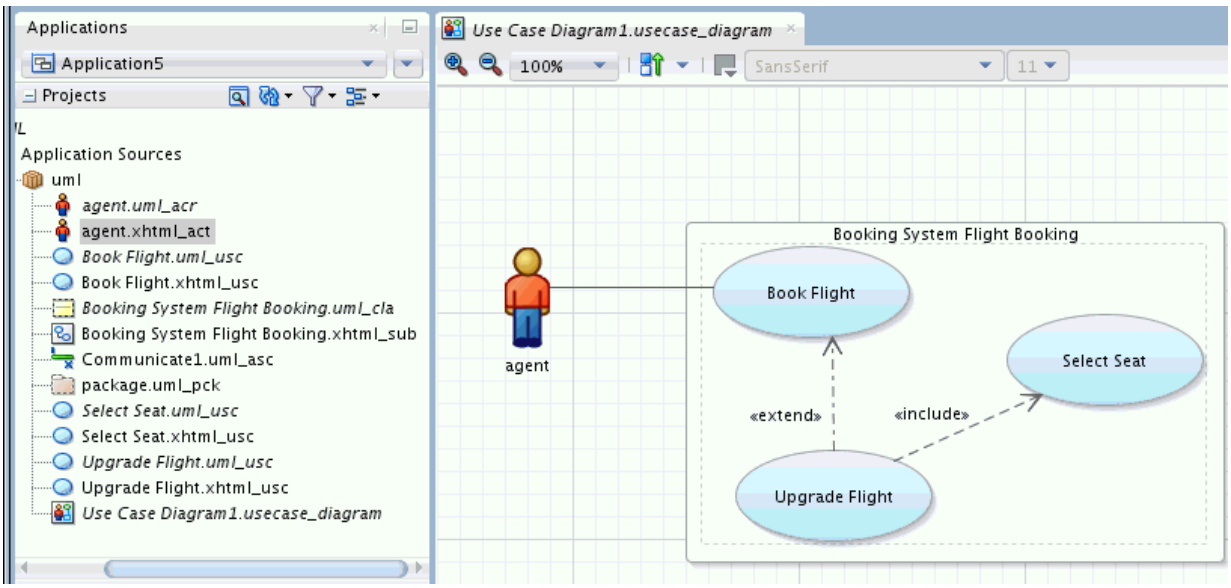
Figure 5-38 Changes to Be Exported

The Exporting Model dialog is a table whose rows represent the details of the changes to be exported. The right-hand column shows the type of model change. There are three types of changes:

- **Added**. The Use Case element does not have a corresponding HTML file yet.
- **Deleted**. The Use Case element has been removed form the model but there is still its corresponding HTML file in the project.
- **Changed**. The Use Case element has been changed in the model and thus it differs from its corresponding HTML file.

The first column of the table contains a check box that enables/disables exporting of the relative model change. Be default, all the check boxes are selected.Once the dialog is confirmed, the HTML files are generated and they will appear in the Application Navigator together with the existing Use Case model elements. The suffixes of the generated HTML files are `.xhtml` rather than `.uml`.

Figure 5-39 Exported Model

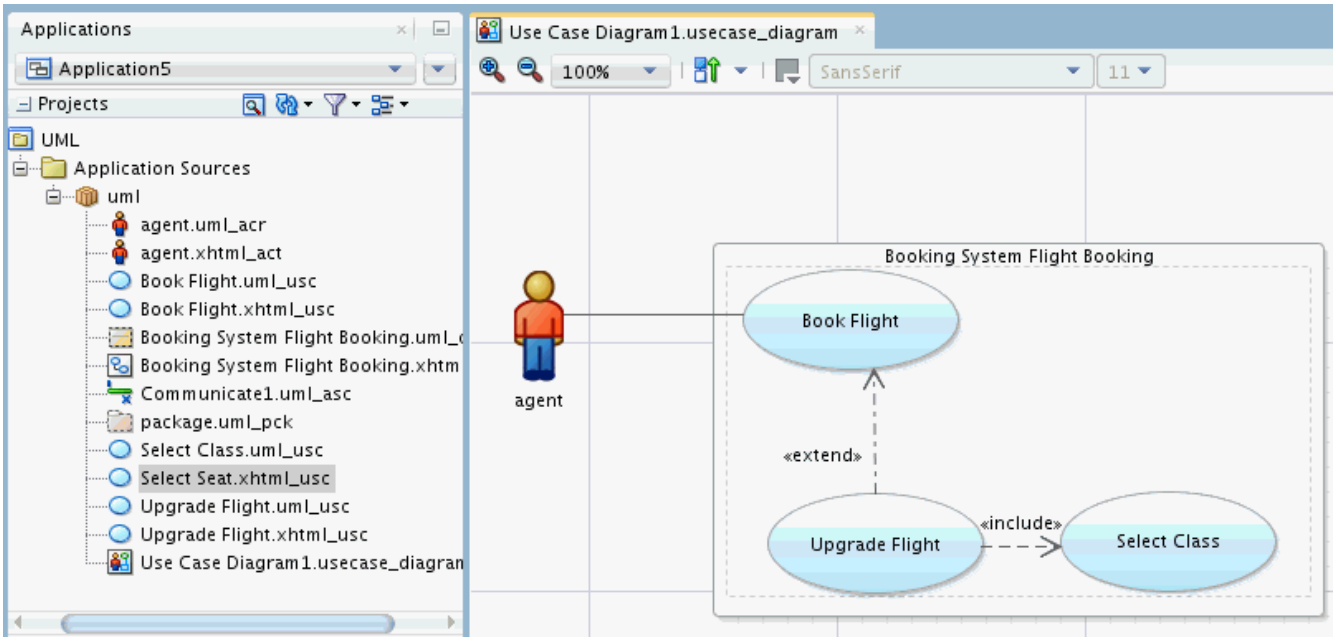


Exporting a Changed Use Case Model

A Use Case model that has been exported can be modified, and the changes sent to an existing set of generated HTML files.

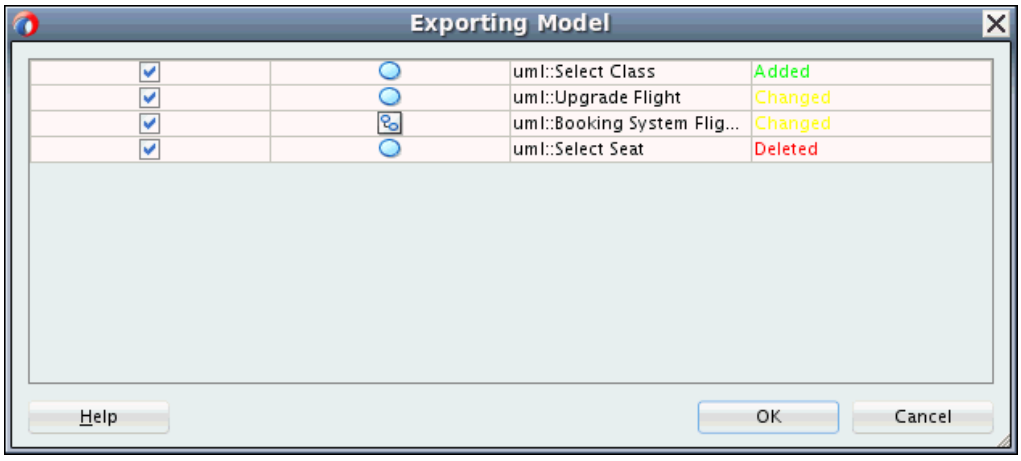
When a Use Case model has been exported it is still possible to modify it and propagate the new changes to the existing generated set of HTML files. Suppose we change Figure 5-40 as shown in Figure 5-41:

Figure 5-40 Use Case Model Modified after Export to HTML



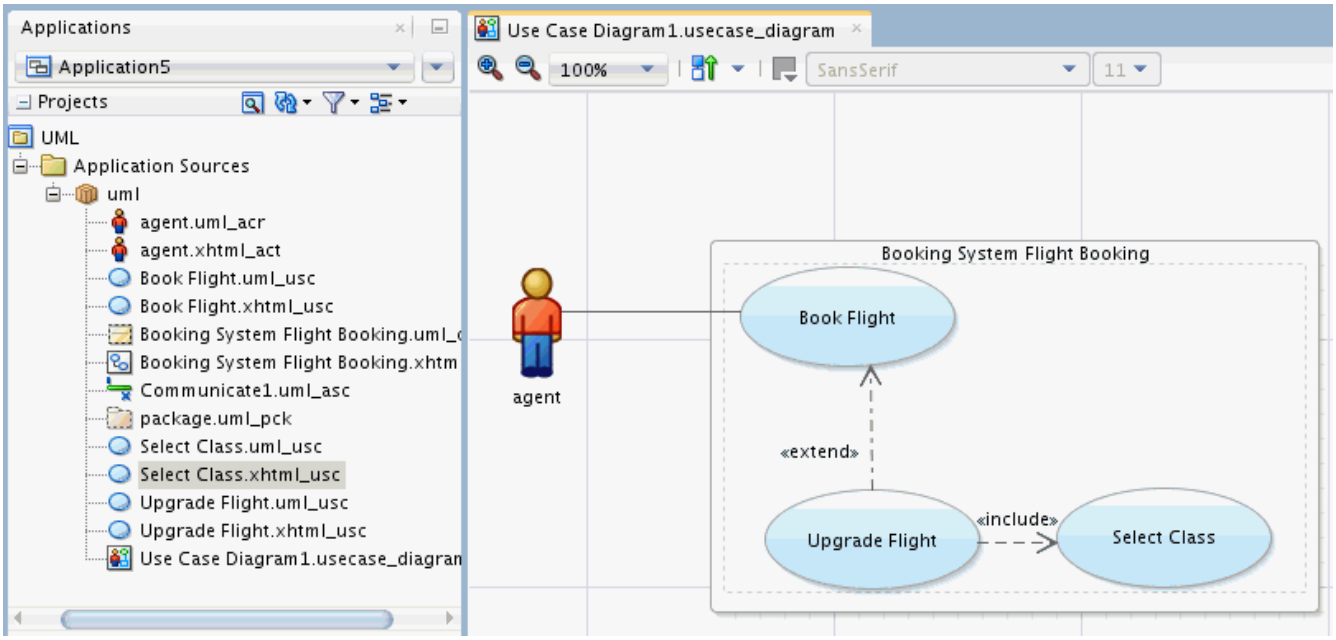
After export the details for the modified model are as shown in Figure 5-41:

Figure 5-41 Exporting a Changed Model



As we can see, the new use case "Select Class" is added to the set of HTML files, while the use case "Select Seat" is removed. The Subject "Booking System Flight Booking" is changed because it contains a new use case, and the use case "Upgrade Flight" is changed because it includes a different use case from before. Once the dialog is confirmed, the changes are applied and they are reflected in the Application Navigator, which now lists the new use case "Select Class" but no longer lists "Select Seat," as shown in the following figure.

Figure 5-42 The Changed Model Following Export



Importing a Use Case Model from a Set of HTML Files

Use the built-in HTML editor in JDeveloper or any other text editor to make minor changes to the HTML files, and apply the changes to the Use Case model. Use Case models can be imported from a set of HTML files.

A Use Case model can also be imported from a set of HTML files that contain the "uml" custom tags. However, creating a Use Case model first entirely in HTML and then importing it into the Use Case modeler it is not recommended. The import functionality exists mainly to allow you to make occasional small changes to the HTML files and apply them to the Use Case model. The HTML files can be edited with the built-in HTML editor in JDeveloper or with any other external text editor.

Editing the HTML Files

HTML files in the Application Navigator can be opened in the HTML source editor for editing. Edit to add associations to or remove associations from use cases.

The HTML files can be edited by double-clicking on their entries in the Application Navigator to open them in the HTML source editor in the main editor panel. The files consist of a HTML document (precisely, XHTML), embedding the UML namespace tags.

Figure 5-43 Editing an HTML Use Case

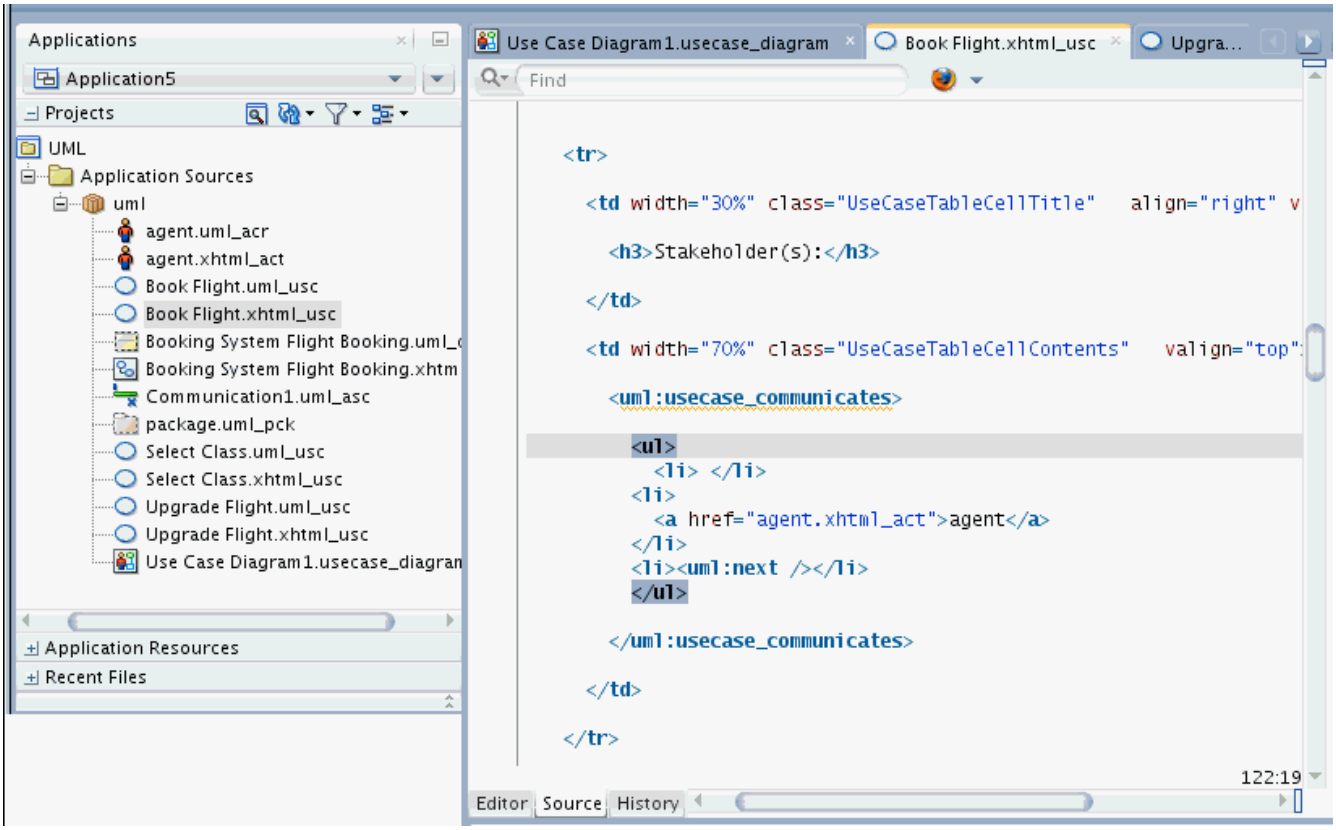
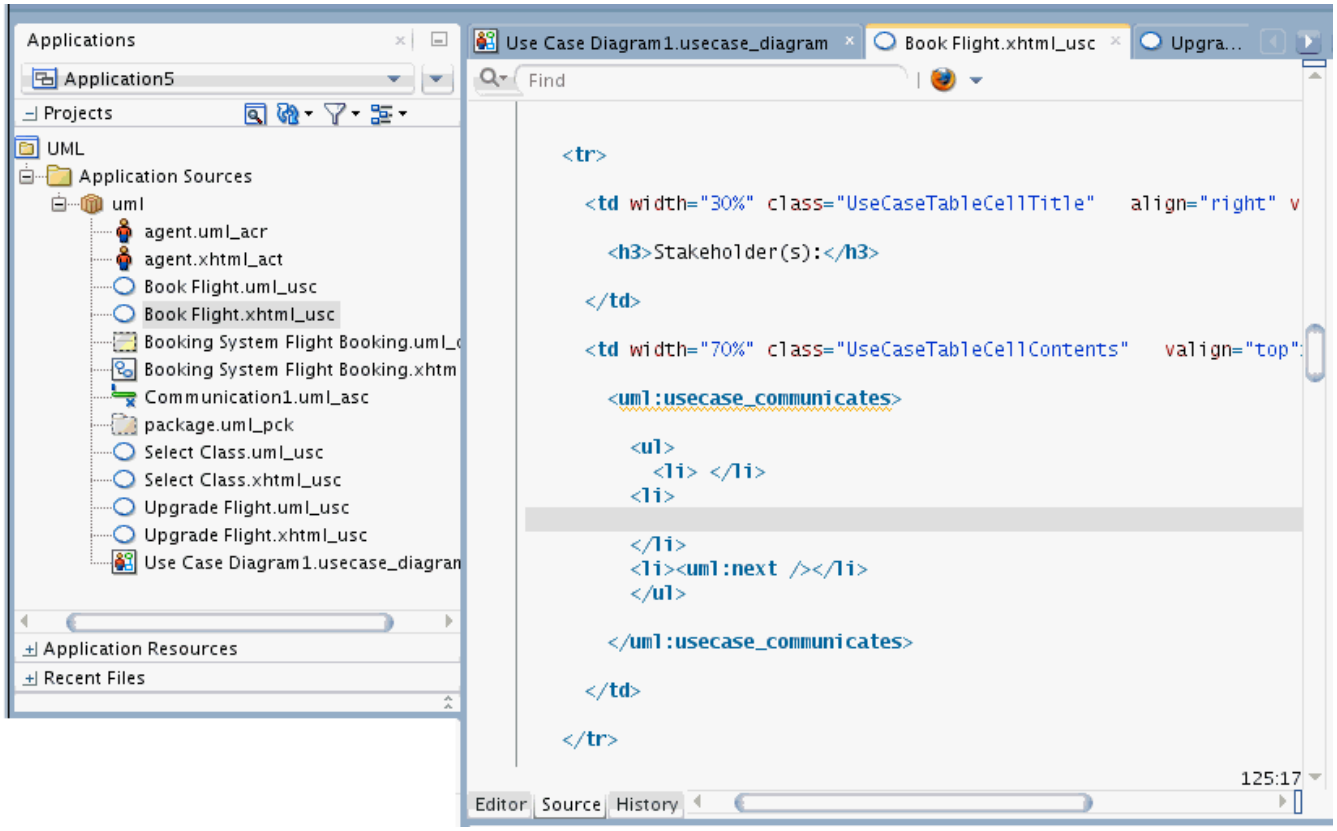


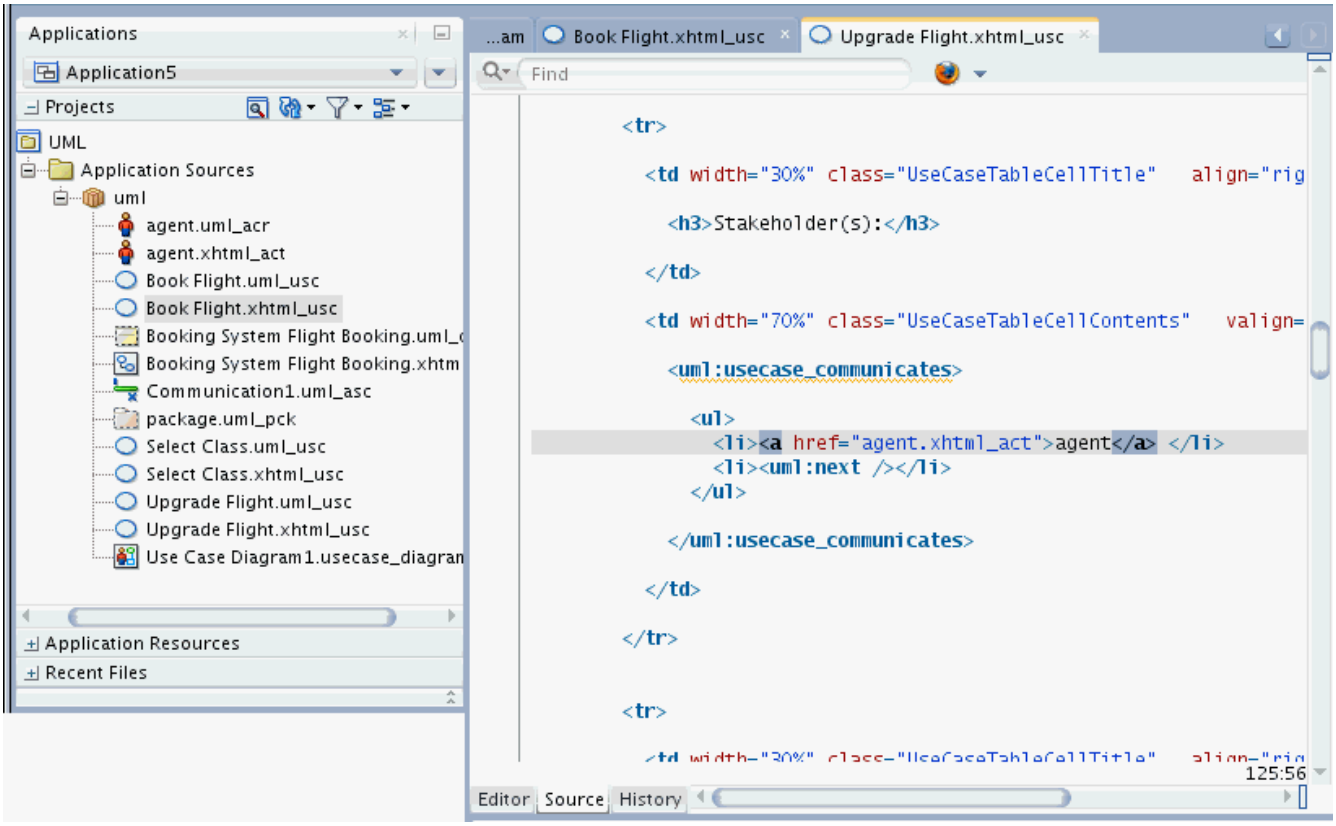
Figure 5-43 shows the editor with the content of a UseCase HTML file. It shows the Communicate association to the relative actor, which is an HTML link contained by the `<uml:usecase_communicates>` element. For example, if we need to remove that associations, we simply remove the anchor element from the enclosing HTML list element, as shown in Figure 5-44.

Figure 5-44 Removing Communication Use Case



To add that Communicate association to a different UseCase, open the HTML files and locate the "`<uml:usecase_communicates>`" element. Then insert the anchor element previously removed into the list element, as shown in the following figure.

Figure 5-45 Adding the Communication Use Case

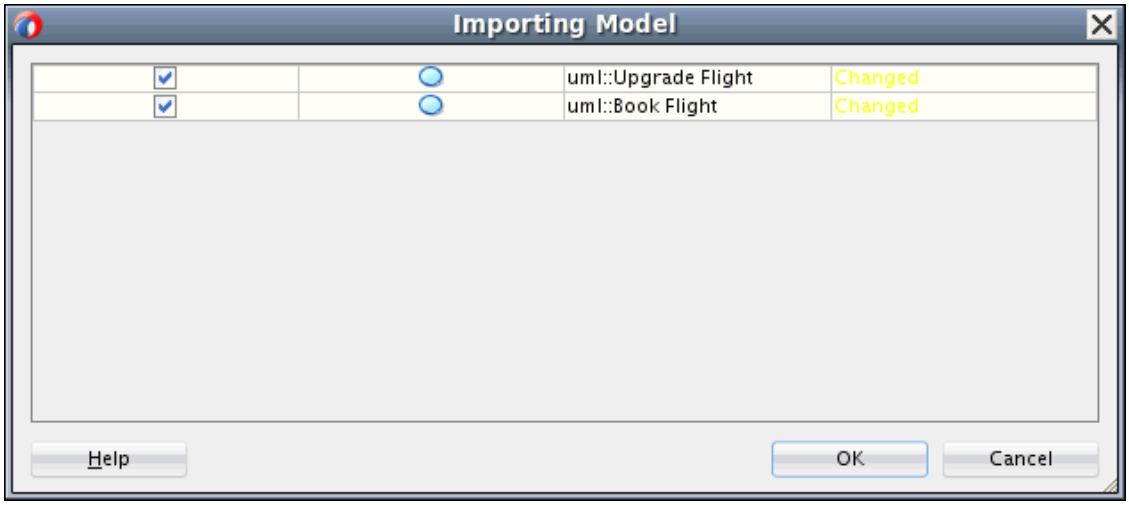


Importing from HTML files

Changes to HTML files can be applied to Use Case elements in models by importing the HTML files. Save the HTML files, and use the Import from XHTML option to import the files.

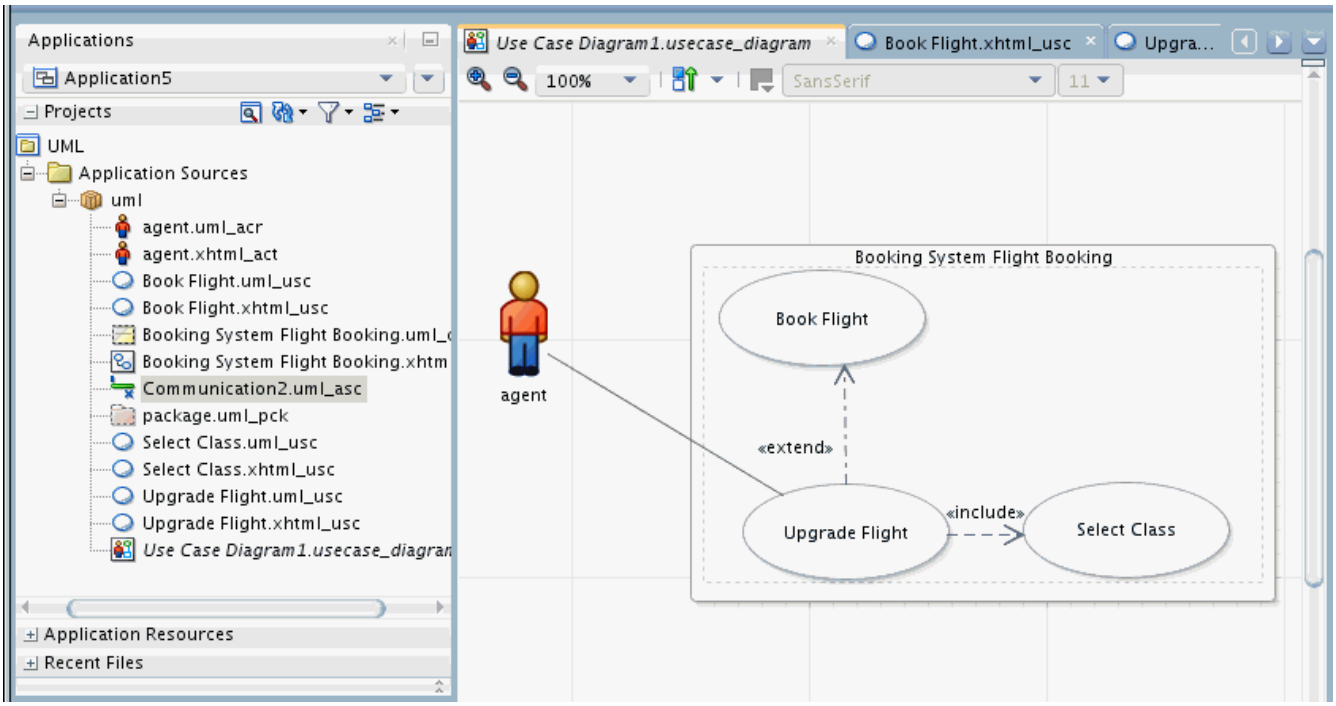
Make sure all the changed HTML files have been saved before performing the import. Right-click on the diagram to display the context menu and choose **Import from XHTML**. The "Importing Model" dialog displays the changes to be applied to the model, as shown in Figure 5-46.

Figure 5-46 Importing a Model



The dialog shows that the changes made to the two HTML files will be applied to the corresponding UseCase elements in the model. Confirming the dialog will cause a new Communicate association to be drawn between the actor and the "Upgrade Flight" UseCase, and between the same actor and the "Book Flight" UseCase removed from the diagram.

Figure 5-47 Revised Model After Import



Modeling with Profile Diagrams

Profiles adapt the UML metamodel for different platforms, domains, or modeled business processes, and multiple profiles can be simultaneously applied to the same model. Use the procedure to create a profile diagram.

Profiles allow adaptation of the UML metamodel for different platforms and domains or your modeled business processes.

The profile diagram is structure diagram which describes the lightweight extension mechanism to UML by defining custom stereotypes, tagged values, and constraints.Stereotypes are specific metaclasses, tagged values are standard meta-attributes, and profiles are specific kinds of packages. Metamodel customizations are defined in a profile, which is then applied to a package.

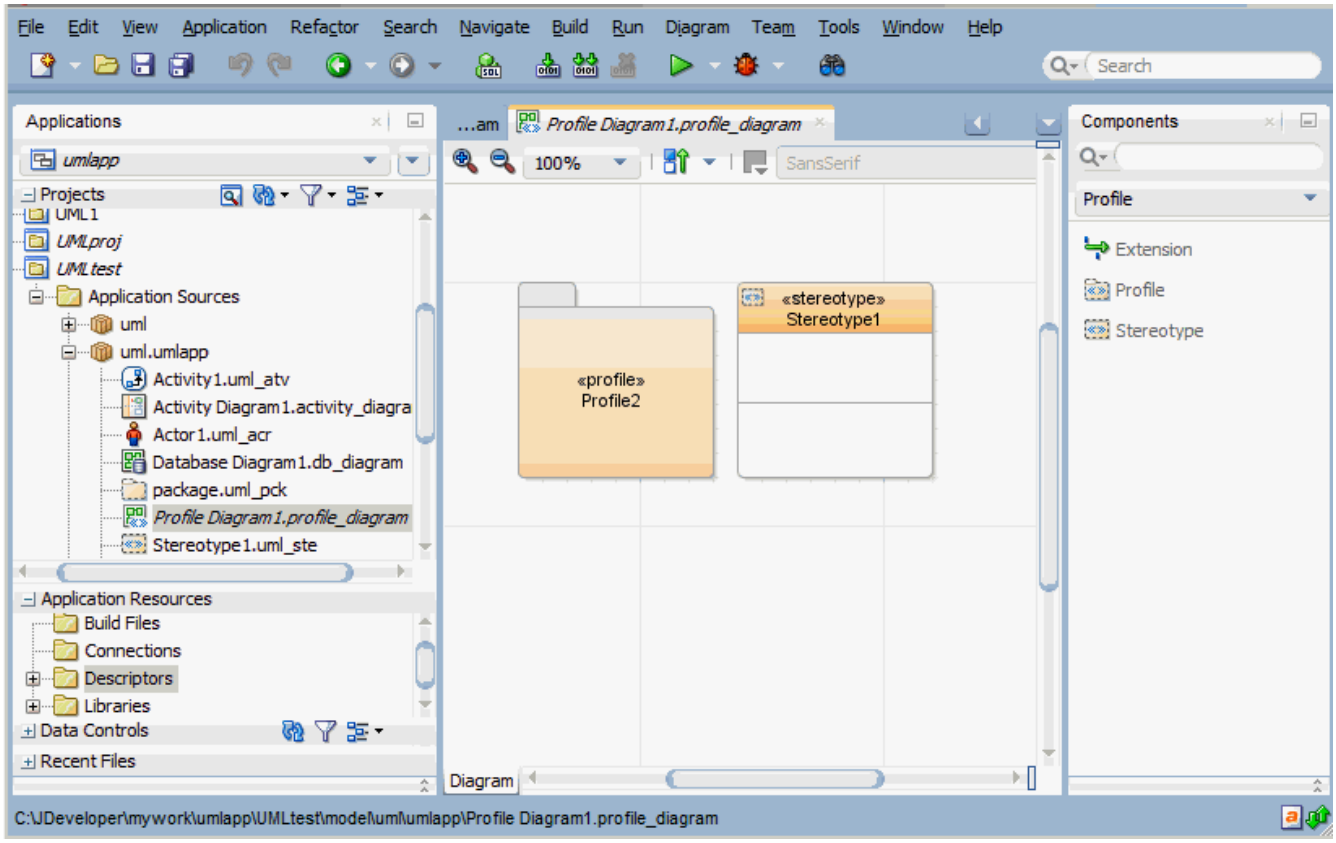
Profiles can be dynamically applied to or retracted from a model. They can also be dynamically combined so that several profiles will be applied at the same time on the same model. Profiles only allow adaptation or customization of an existing metamodel with constructs that are specific to a particular domain, platform, or method. You can't take away any of the constraints that apply to a metamodel, but using profiles, you can add new constraints.

How to create a profile diagram:

1. Select **File > New**. The New Gallery opens.
2. In the Categories panel, open the General node and select the UML node. The UML elements are listed in the Items panel.
3. In the Items panel, select Profile Diagram, and click **OK**.
4. Supply the profile diagram details and click **OK**. The profile diagram is added to the applications window.

See Using UML Profiles for instructions on how to export a profile as XMI, add a profile to a diagram, and apply a profile to a UML package.

Figure 5-48 Profile Diagram



Modeling with Java Class Diagrams

A Java class diagram can contain shapes from other diagram types. As the Java source code provides definitions of the classes on a diagram, diagrams reflect changes to source code, and source code reflects changes to diagrams.

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).

How to Create Java Classes, Interfaces and Enums

In the Java Components window for the diagram Java Class, Java Interface, or Java Enum, create Java classes, interfaces, or enums. The project settings specify the location for the creation of the Java source file for the modeled class or interface.

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 Components window 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.

How to Model Inner Java Classes and Interfaces

The inner classes compartment of the modeled Java class or interface displays inner classes and inner interfaces. Create inner classes or inner interfaces by adding 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.

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.

Modeling Composition in a Java Class Diagram

The source code of the implementing classes of the references shows references created between modeled Java classes as fields. Create references between classes and interfaces on a diagram using the Java Class Components window for the 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 Components window 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 5-10 displays the references that can be modeled on a diagram.

Table 5-10 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.

Modeling Inheritance on a Java Class Diagram

Use Extends in the Java Class Components window for a diagram to create `extends` statements in the Java source to represent inheritance structures. If a class implements an interface, use Implements in the Java Components window for the diagram to create inheritance structures.

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 Components window 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 Components window 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.

Extending Modeled Java Classes

Extends relationships show inheritance between elements in a class model. Extend relationships between Java classes and between Java interfaces can be created, but enums can neither extend other classes, nor be extended by other 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.

Implementing Modeled Java Interfaces

Implements relationships specify where a modeled Java class implements a modeled Java interface. A class that implements an interface can provide an implementation for all the abstract methods of the interface, and a class that partially implements the methods of an interface is defined as abstract.

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 can

not 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.

Modeling Java Fields and Methods

Fields encapsulate the characteristics of a modeled Java class or Java interface, and methods in a class define the behavior of the class. Create fields and methods for modeled Java classes and interfaces on a diagram, and add them using the Java Source Editor.

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.

Refactoring Class Diagrams

Changes to Java class diagrams are refactored in source code by means of refactoring patterns for the Java fields and methods such as Rename, Move, Duplicate, Extract Interface, and Extract Superclass. Use the procedure to invoke a refactoring operation.

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:

To invoke a refactoring operation:

1. Select a program element in a source editor window, databases window, 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:
   - Rename
   - Move
   - Make Static
   - Pull Members Up
   - Push Members Down
   - Change Method (Java methods only)

Modeling with EJB Diagrams

Create an EJB diagram, and add elements such as UML classes, Java classes, and business components to Model EJBs. Modeling options include Container-Managed Persistence, Bean-Managed Persistence, sessions whose session type can be changed on a class diagram, and message-driven beans that are used to interact with session and entity beans.

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. 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.

The following are the modeling options available:

- 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.

Working with EJB/JPA Modeling Features

Use the Entity Bean, Session Bean, or Message-Driven Bean components in the Components window to create Enterprise JavaBeans on a diagram. Add properties and methods using the EJB Module Editor, by creating the new property, or method 'in-place' on the modeled bean itself.

Enterprise JavaBeans are created on a diagram by using the **Entity Bean** icon, **Session Bean** icon or **Message-Driven Bean** components on the Components window. Select the component and then click the diagram in the desired spot. The implementation files for the modeled elements are created in the location specified by your project settings.

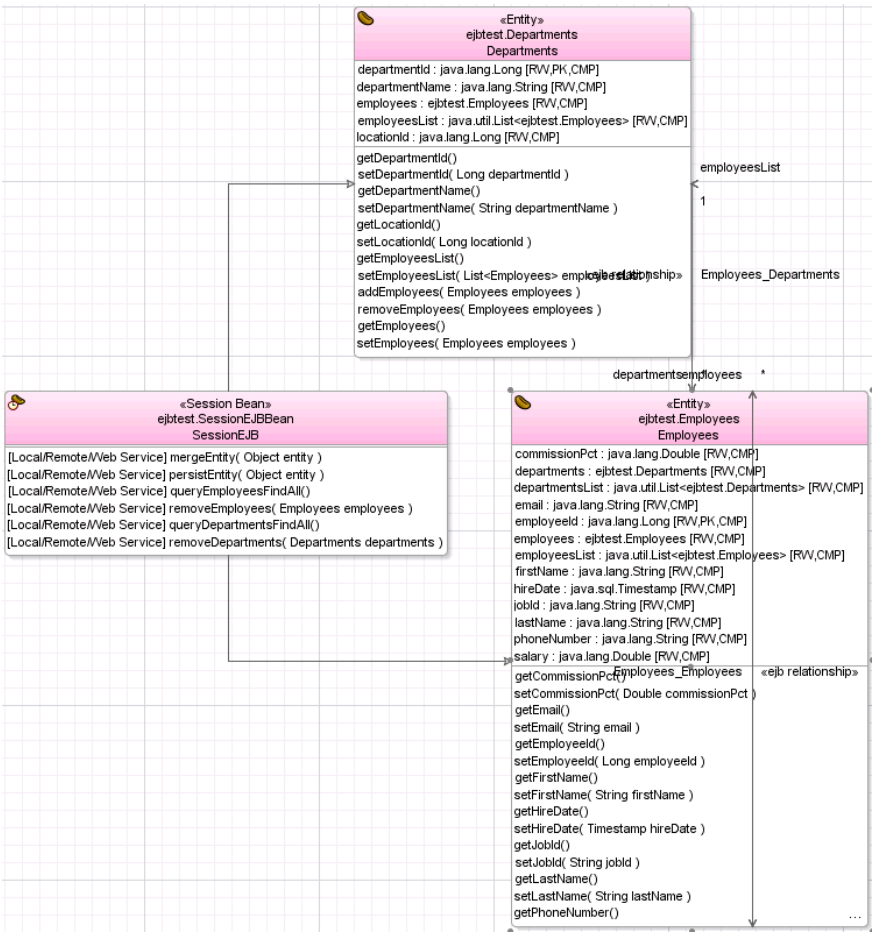
> **Tip:**
>
> If you want to model the implementing Java classes for a modeled bean on a diagram, right-click the modeled bean and choose **Show Implementation Files**.

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.

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 5-49 EJB/JPA Components Diagram



Notice the relationship and edges between the beans. References can be created from any bean to another bean with a remote or local interface. References can only be modeled between beans that are inside the current deployment descriptor.

Create a Diagram of EJB/JPA Classes

Use the procedure to create a diagram of EJB or JPA classes.

To create a diagram of EJB/JPA 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 Components window. Table 5-11 shows the EJB Components window.

Table 5-11 EJB Components window 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 Inheritance Edge
		Session Facade Edge
Diagram Annotations		Attachment
		Group
		Link
		Note

How to Model EJB/JPA Relationships

Model a relationship between any two entities on a class diagram using the Components window. Use the procedure to model a relationship between two entities on a diagram.

You can model a relationship between any two entities on a class diagram by dragging the relationship component from the Components window. You can also show the inheritance edge between the root and child entity.

To model a relationship between two entities on a diagram:

1. Click the icon for the relationship to create.

> **Note:**
>
> The navigability and multiplicity of a relationship end can be changed after it has been created.If EJB component icons are not displayed, select **EJB Components** from the dropdown on the Components window.

2. Click the entity at the 'owning', or 'from', end of the relationship.
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 > ***.

Reference Between Beans

Relationships can be created between modeled entity beans. The EJB Reference option creates references from a bean to another bean with a remote interface, and the EJB Local Reference option in the EJB Components window for the diagram creates local references from a bean to another bean with a local interface.

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 Components window for the diagram.

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.

Properties on Modeled Beans

Add properties to modeled EJBs using the EJB Module Editor or by creating the new property or method directly on the modeled bean.

Properties 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.

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

Methods on Modeled Beans

Create either local and remote, and local and local home methods on modeled beans on a class diagram.

Both local/remote and local/local home methods can be created on modeled beans on a class diagram.

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.

> **Note:**
>
> 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 applications window, then change the details of the method using the EJB Editor.

How to Model Cross Component References

EJB References and EJB Local References can be created between modeled beans on a class diagram. Use the procedure to model a reference between modeled beans.

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 from those listed on the EJB Components window:
   - EJB Reference
   - EJB Local Reference
2. Click the bean at the 'owning', or 'from', end of the reference.
3. Click the bean at the 'to' end of the reference.

How to Display the Implementing Classes for Modeled Beans

The Java source files that contain the implementation code for a modeled bean are displayed on the diagram as modeled Java classes. Follow the steps to display a modeled implementing Java class 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 you want to model on the diagram, then choose **Model** > **Show** > **Implementation Files**.
- Or, right-click the bean and choose **Show Implementation** > **Files**.

How to Display the Source Code for a Modeled Bean

JDeveloper displays Java source code for a modeled bean in the source editor with simple commands on the diagram. Follow the steps to display the Java source code for a model element.

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.
- Select the element and choose **Model** > **Go to Source**.

How to Change the Accessibility of a Property or Method

Change the accessibility of a property or method using the context menu. Use the procedure to change the accessibility of a property or method.

You can change the accessibility of a property or method using right-click.

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.

   The accessibility options are:

   - Local Interface
   - Remote Interface
   - Local and Remote Interfaces

How to Reverse Engineer a Diagrammed JPA Entity

Reverse-engineer table definitions in the application database connection to modeled entity beans on a diagram of EJBs. Use the procedure to reverse-engineer a table definition to an entity bean.

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 window 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 databases window and drag these tables to the diagram, then release the **Ctrl** key.
5. Select the EJB version and click **OK**.

Modeling with Database Diagrams

A model of database structures provides a diagrammatic view of database schema and the relationships between online and offline tables.

Modeling your database structures gives you a visual view of your database schema and the relationships between the online or offline tables. You can also transform database tables to UML classes and vice-versa using the transformation features. For information about database transformation see UML-Offline Database Transformation.

Working with the Database Modeling Features

Use JDeveloper to model offline database objects and database objects from a live database connection. Database diagrams provide structural views of database objects and relationships, and the ability to create components on a diagram.

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. All of the database objects from online or offline databases, as well as the new objects you create are displayed in the Applications window.

- Offline objects appear in the Application Navigator under the "Offline Database Sources" node.
- Online database connections that are part of your application appear in the Application Navigator Application Resources view under the Connections > Database node.
- Online database connections that are shared across applications appear in the Database Navigator which can be accessed from Window > Database > Databases.

Use database diagrams to view your structure of database objects 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.

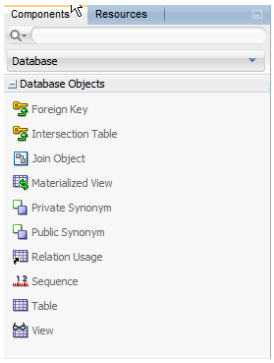
How to Create a Database Diagram

Create a database diagram using New Gallery, and add components to it.

Create your database diagram using the New Gallery. See Creating a New Diagram.

Once your database diagram is created, you can choose from the components in the Components window, as shown in Figure 5-50.

Figure 5-50 Database Components Window



How to Create an Offline Database Object

Create an offline database object on a diagram using the Database Objects Components window. Dragging objects from a database connection in the Databases window, or from an offline schema in the Applications window also creates database objects.

To create an offline database object on the diagram, click on the icon on the Database Objects Components window, and then click on the diagram where you want to create the object. This process adds existing objects to a database diagram.

You can also drag objects from a database connection in the Databases window, or from an offline schema in the Applications window.

How to Create a Foreign Key

Click the table containing the foreign key, and then click the target table to create a Foreign Key in a database diagram.

To create a Foreign Key in a database diagram, click the table from which the foreign key originates, and then click the target destination table. The Create Foreign Key dialog allows you to select an existing column in the target table, or create a new column. The target table will be the owner of the foreign key.

How to Use Templates to Create Database Objects

Use the <`ObjectType`> component to create an object based on a template that is defined in the Offline Database Properties dialog.

All templates are defined in the Offline Database Properties dialog. If the object being created has a template defined in the offline database in which it is being created, then using the `<ObjectType>` component creates objects based on its template. For example, if the offline database 'Database1' has a Template table 'MyTab', when you create a new table in Database1 using the table option in the Component window, the new table created is based on Template 'MyTab'. The existing objects from the offline database are added to the diagram.

How to Add and Create Private and Public Synonyms

Select a Database from the list in the Components window and use the Public Synonym or Private Synonym options to create Synonyms on a diagram. Alternatives include adding objects from an online database connection, and importing objects from the database connection to the offline database for addition to the diagram.

To create Synonyms on a diagram, go to the Components window and choose Database. From the Database list, click Public Synonym or Private Synonym and then click the diagram in which you want to create the synonym. Note, both components contain a checkbox for the Public property. The box is automatically checked for the Public Synonym, and it is unchecked by default for the private synonym.

You can also drag+drop objects from an online database connection. To import objects from the database connection to the offline database, add the existing objects from the database connection to the diagram.

How to Add and Create a Sequence

Use the Sequence option in the Database Objects components window for the diagram to create a sequence on a diagram. Sequences can also be added from a database connection, or from an offline database in the Applications window.

To create a sequence on a diagram, go to the Database Objects components window for the diagram, and select Sequence, then click on the diagram where you want to create the sequence.

You can also drag sequences from a database connection, or from an offline database in the Applications window, and drop them on the diagram.

How to Add and Create Tables

Use the procedure to add tables to a diagram.

Follow these steps to add tables to a diagram.

1. Go to the Database Components window, click on Table, and then click on the diagram to insert the table.

   You can also drag tables from a database connection or an offline database in the Applications window, and drop them on the diagram. Existing objects from an offline DB can be added to the diagram in the same way.

   To display various attributes on a modeled table, select **Preferences > Diagrams > Database**, and from the "Edit Preferences for" dropdown, select Table. On the Display tab, check Use Tabular Layout and Show Icons. To view table constraints, select the Constraints tab and check any constraint attributes you want to display.

   The first column in the modeled table indicates whether the column is in a primary, unique, or foreign key. 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.

2. To further define the table, click on the table and choose Properties from the context menu.

   On the left is a list of the properties you can define. At the bottom of the panel note that the Overview tab is the default view. To change the view, click the Diagram, DDL, or History tabs.

3. To add columns, choose Columns on the left. In the Columns table, click the + button to add a row for each column you require. For each new column use the tabbed panel to specify the column's Data Type, Constraints, Indexes, LOB Parameters, Identity Column, and User Properties.

4. To add constraints, choose Constraints on the left, and use the + dropdown menu to specify new keys or constraints. Provide the Type and Name and set the deferrable state. With the constraint selected in the Constraints table, use the tabbed panel to specify its Properties and User Properties.

   To apply a constraint to a column, select the constraint in the Constraints table, and in the Properties tab below, choose an index, then from the Available Columns area, choose a column and click the > button to move that column to the Selected list.

How to Change the Database or Schema

Use the procedure to change the database or schema. Database objects created subsequent to the change will be in the selected 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 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.

How to Create Database Views and Add Database Objects

Use the procedure to create a database view. Importing objects from a database connection to an offline database adds objects from the database connection to the diagram.

Follow these steps to create a database view:

1. Open the diagram that needs the database view.
2. In the Components window Database category, expand Database Objects and select the View icon.
3. Click in the diagram. The View Object is added to the diagram.

   To define the view add tables and views from the palette, or table columns or elements of other views.

   To add an existing view, you can drag views from a database connection and drop them on the diagram. You can also drag in views from an offline database displayed in the Applications window.
4. Right-click the view component and select Properties. Specify the schema and name, and fill out the SQL Query and Properties forms.

To import objects from a database connection to an offline database, add existing objects from the database connection to the diagram.

How to Define a Base Relation Usage

A FROM clause comprises a component called base relation usage that specifies a relation to a table or view. Use Database from the Components window to define a base relation for a view.

The base relation usage is a component of a FROM clause that specifies a relation to a table or view. When you define an SQL Query with a FROM clause specified (you can type it in or use the Query Builder) a Relation Usage object is automatically added to the View component.

To define a base relation for a view, go the Components window and choose Database. In the list, click on Relation Usage and then click on the View shape.

How to Create Join Objects

Using Join Object you can create join objects between two table usages in a view.

To create join objects between two table usages in a view, click Join Object, then click on the two table usages to be joined. The Edit Join dialog allows you to specify the join.

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:

- About Versioning Applications with Source Control
- Downloading Source Control Extensions in Oracle JDeveloper
- Setting Up and Configuring Source Control
- Setting Up and Configuring a Source Repository
- Working with Files in Source Control
- Working with Branches and Tags
- Working with File History, Status and Revisions
- Working with Patches in Source Control

About Versioning Applications with Source Control

Version control and source control help teams coordinate work. Version control tracks application changes against project management or bug reporting systems, and developers check in code into a commonly used repository of content that is built into a functioning product.

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.

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.

Downloading Source Control Extensions in Oracle JDeveloper

Integrate a version control system with JDeveloper and download a JDeveloper Extension using Check for Updates from Help. Downloadable extensions are available for Mercurial, Perforce, and Microsoft Team System.

For users familiar with other versioning systems, or whose teams use systems other than Subversion and Git, JDeveloper provides downloadable extensions that give you access to the following:

- Mercurial
- Perforce How to Set Up Perforce with JDeveloper

- Microsoft Team System How to Set Up Team System and JDeveloper

If your team requires 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.

Setting Up and Configuring Source Control

Besides integrated solutions such as Subversion and Git, JDeveloper supports extensions such as Mercurial, and an application lifecycle management system. Access commands for the systems from the Team menu or from the Versions window.

JDeveloper offers several tools for developing in teams. These include integrated solutions such as Subversion and Git, as well as downloadable extensions such as Mercurial. You can access commands for all of these systems directly from the JDeveloper interface, through the **Team** menu or through the Versions window

Setting Up Subversion and JDeveloper

Set up Subversion by creating a repository for source-controlled files, establishing JDeveloper connectivity to that repository, and importing files to the repository. Check files out from the repository, edit them in JDeveloper, and check them in with changes.

JDeveloper is integrated with the popular team development solution Subversion (SVN). If you are part of a team that uses Subversion, JDeveloper's Team 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." 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.

Installing Subversion Client Software

Install Subversion client software if a local Subversion repository is needed, if a Java binding other than SVNKit is to be used, and if a proxy server is used to connect to the Subversion repository. Use the procedure to install Subversion client software.

In addition to creating a repository for your source-controlled files, making sure that JDeveloper can connect to that repository, and importing files to the repository, it may be necessary to install Subversion client software under 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.

To install Subversion client software:

1. Download the Subversion installer (`svn-1.7.8-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`).

Checking the Subversion Client Installation

Verify whether the Subversion Client that has been installed is available. Use the procedure to check the installation.

Once you have completed installing the Subversion client software, you can check the Subversion Client installation.

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.

To check the installation:

1. In JDeveloper, select Subversion as the versioning system (**Team > Configure**, and then select **Subversion**).
2. 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.

Creating a Subversion Connection

Connect JDeveloper to Subversion to work with a Subversion repository through JDeveloper. Use the procedure to create 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 Versions window (**Team > Versions**), 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**.

Editing a Subversion Connection

You can edit details such as the IP address, port, user ID, or password of a Subversion connection. Use the procedure to edit a Subversion Connection.

If any of the details (such as IP address, port, user ID, password, etc.) of your Subversion connection change, you can edit the connection in JDeveloper so that you can connect to it with the new details.

To edit a Subversion Connection:

1. In the Subversion Navigator (**Team > Versions**), 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**.

Exporting Subversion Repository Connection Details

Exporting the details of Subversion repository connections to a file helps in recreating Subversion repository connections. Use the procedure to export Subversion connection details to a file.

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. This can greatly simplify the process of connecting to a Subversion repository as new team members join, as the repository connection file can be stored on a server accessible to the team, then downloaded as required as new members join. Similarly, if new servers are added, the team leader or administrator can distribute new connection information in a connection detail file to be imported by all team members.

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.

Importing Subversion Repository Connection Details

Importing information of a Subversion repository connection into JDeveloper simplifies connecting to a repository. Use the procedure to import Subversion connection details from a file.

If you or your team have saved the details of your Subversion repository connection, you can import them into JDeveloper to simplify making the connection to your repository.

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**.

Connecting to a Subversion Repository Through a Proxy Server

For connections to a Subversion repository through a proxy server, install the Subversion client software, edit the `servers` file in the Subversion subdirectory, and ensure that the proxy server supports all the `http` methods used by Subversion.

If you want 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.)

> **Note:**
>
> If you have entered the proxy settings in the JDeveloper Preferences, you can omit editing the servers file as described in the following paragraphs.

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.

Exporting Subversion Controlled Files from the Working Copy

Export copies of JDeveloper files that are under Subversion control from the Applications window, or from the Subversion Navigator. Use the procedure to export files from the Applications window.

You can export copies of JDeveloper files that are under Subversion control from either of two places: the Applications window, in which case the files will be exported from the Subversion “working copy", or the Subversion Navigator, in which case the files will be exported from the Subversion repository. Exporting the files means copying them to a local file system directory that you specify.

To export files from the Applications window:

1. In the Applications window, select the project containing the files that you wish to export.
2. Select **Team > 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**.

This exports the selected files from the Subversion "working copy" of your local file system.

Exporting Files from the Subversion Navigator

To export a specific revision of files, export copies of the files under Subversion control from the Subversion Navigator. Use the procedure to export files from the Subversion Navigator.

You can export copies of files under Subversion control from the Subversion Navigator. This ensures that the files will be exported from the Subversion repository (not from the "working copy"). In addition, exporting with the Subversion Navigator lets you specify which revision of the files to export.

To export files from the Subversion Navigator:

1. In the Subversion Navigator, select the repository node or directory containing the files that you wish to export.
2. Select **Team > 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**.

This exports the selected files from the repository to your local file system.

How to Set Up and Configure a Git Repository

To use the open-source version control system Git with JDeveloper, clone the team repository on a local system, and ready information such as repository name, URL, and credentials for access. Use the procedure to connect to Git.

Git is a popular open-source version control system with a growing user community. To begin using Git with JDeveloper, you create a clone of your team's repository on your local system.

To complete this procedure, you will need the following information, which should be available from the Git administrator on your team:

- The name of the repository
- The URL at which the repository is stored
- The user name and password you will use for accessing the repository. Optionally, your team may use a private key file with a passphrase; you can select the appropriate option for your team when you connect to Git.
- The name of the remote branch that your team uses for development.
- The destination pathname in your local system to which you wish to store your local repository.

To connect to Git:

1. Select **Team > Connect to Git**. This displays the Clone from Git wizard welcome screen. Click **Next** to continue.
2. Enter the information about your team's remote Git repository, then click **Next** to continue.
3. Specify the remote branch your team uses for development, then click **Next** to continue.
4. Specify the pathname on your local system at which you wish to create your local repository, then click **Next** to continue.
5. Verify all the information displayed in the Git Clone Summary screen, then click **OK**.

JDeveloper connects you to the remote repository and creates a local clone based on the branch you selected. From this local clone you can check out, edit, merge, and commit files to the main repository.

How to Set Up Perforce with JDeveloper

JDeveloper requires that the Perforce extension and Perforce features be installed for it to work with Perforce. JDeveloper must be configured to connect to the Perforce client workspace, and working files must be brought under Perforce control.

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.

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.

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.

Installing Perforce Components for use with JDeveloper

For JDeveloper to work with Perforce, at least one Perforce server must be installed on a machine that is accessible to JDeveloper users.

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.

Perforce Client Installation

The Perforce client application and the Command Line Client must be installed on the same computer as JDeveloper. Create a Perforce client workspace which stores working copies of files under Perforce control.

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)".

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.

If you set up passwords in the Perforce client application, you will also need to use them when connecting to Perforce through JDeveloper.

JDeveloper Installation

Install JDeveloper on every computer that must function as a Perforce client. JDeveloper functions as a client application for Perforce.

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.

Configuring JDeveloper for Use with Perforce

Install the Perforce server and client software, and configure JDeveloper to use Perforce. Use the procedure 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 > Versioning > Perforce**.
2. Verify that the path to the Perforce client is as you installed it.
3. If your team uses comment templates, select Comment Templates from the left-hand pane and configure your team's comment templates.
4. In the left pane of the Preferences dialog, select General, then make selections about icons, log messages, opening files automatically for edit, and the length of the idle period before timeout.
5. In the left pane of the Preferences dialog, select General, then make selections about the Pending Changes window and the Merge Editor.
6. To save the configuration you have just set up, click **OK** to close the Preferences dialog.

Selecting Perforce as the Version System

After configuring JDeveloper for use with Perforce, specify Perforce as the version control system so that all team operations default to Perforce. Select the Perforce option available on the Team menu to specify Perforce as the version system.

Once you have configured JDeveloper for use with Perforce, you can select Perforce as the version system. This will specify a number of default settings which mean that all team operations will default to Perforce as your chosen version system.

To select Perforce as the version system:

- Choose **Team > Perforce**.

You can change this default selection at any time if your team changes to a different version system at a later date.

How to Set Up Team System and JDeveloper

Use the source control features of Microsoft Visual Team System within JDeveloper with the Oracle JDeveloper Team System extension. Use the steps to set up the team system client software and configure 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 Applications windows.

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.

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 Setting 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 Configuring 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 Checking Out Files in Team System.
- Making changes to a file saved in your Team System workspace, and make them available to other users. See How to Check In Files to Team System.
- Using Team System's Shelving feature to save file changes in the Team System server without having to check the files in. See Shelving and Unshelving Team System 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.

Setting Up Team System for Use with JDeveloper

Install the software, connect to the server, and populate the workspace to use Team System with JDeveloper. Use the procedure to set up Team System with JDeveloper.

Using Team System with JDeveloper requires a setup procedure that includes installing software, connecting to your server, and populating your workspace.

To set up Team System with JDeveloper:

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.

Configuring JDeveloper for Use with Team System

Install JDeveloper Team System VCS extension and configure JDeveloper. Use the steps 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 Setting Up Team System for Use with , make sure you have already 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:

- Connect to 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.

Selecting Team System as the Versioning System

Connecting to Team System ensures that it becomes the target for tasks from the Team menu. Use Connect to Team System available on the Team menu to connect to Team System.

Connecting to Team System as the default versioning system specifies that Team System is the target for a number of actions from the Team menu.

To connect Team System as the versioning system:

- Choose **Team > Connect to Team System**.

This displays the Team System connection menu, from which you can select the available operations. Detailed instructions for this are given in the Team System online help.

Setting the Team System Workspace to use JDeveloper

Set up the selected Team System workspace to use JDeveloper. Use the procedure to set the workspace to use with JDeveloper.

Before beginning, you need to set your selected Team System workspace to use JDeveloper.

To set the workspace to use with JDeveloper:

1. Choose **Team > Team System > Set Workspace**.
2. Select the required workspace from the list.

Creating a JDeveloper Project for the Workspace Files

When you create and edit files, the JDeveloper project and the selected Team System workspace association keeps the work in the team workspace. Use the procedure to create a JDeveloper project to hold the workspace files.

Associating the JDeveloper project with the selected Team System workspace ensures that the files you create and edit will remain part of the workspace your team is using.

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 Applications window, 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.

Once completed, refresh the workspace folders in JDeveloper by choosing **Team > Refresh Workspace Folders**.

Getting Versions of Files from the Team System Server

From the Team System server, you can get different versions of files. Use the procedure 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 Applications window. 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 Applications window.

To get versions of files from the Team System server:

1. In the Applications window, select the application, project or files to set the scope of the Get operation.
2. Select **Team > Team System > Get.**

   The Get dialog is opened.
3. Complete the dialog.

   For information while using the dialog, press **F1**.

Adding Files to Team System Control

When files are brought under Team System source control, the files are added to the Team System server and made available to other users when they are checked in. Use the procedure to add files to Team System Control.

You can 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 Applications window, select the file that you want to add to Team System control.
2. Select **Team > 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.

Versioning Applications With Mercurial

Mercurial is a Source Control Management system that works with distributed repositories. Mercurial commands can be run on both files and directories in the Projects, Files and Favorites windows.

Mercurial is a Source Control Management system designed for efficient handling of very large distributed projects. Unlike Subversion, Mercurial works with distributed repositories which are commonly used in many open source projects today and support distributed development without any centralized control.The Mercurial Plugin support enables you to manage changes to version-controlled files as you work. You can call Mercurial commands on both files and directories in the Projects, Files and Favorites windows.

The advantages of a distributed revision control system like Mercurial include:

- Better support for distributed teams by removing a centralized bottleneck
- Better scalability with large numbers of concurrent users
- After the initial clone, faster to work with, independent of a user's network infrastructure

About Mercurial Visualization Features

JDeveloper provides several file status information tools, such as Color Coding and Annotations, to simplify the process of working with version-controlled files.

Color Coding enables you to view the current status of version-controlled files. Annotations enables you to view revision and author information for each line of version-controlled files.

JDeveloper's Mercurial support is similar in style to its Subversion support. The main difference is that Mercurial is a distributed revision control system. Therefore, you typically begin by cloning an external repository to work with. This clone is a complete copy of the repository including the revision history. You can clone this local copy as often as you like. When you want, you can push your changes back to the original repository provided you have permissions, or export your changes and send them to the owner if you do not.

For more information on Mercurial Plugin support and Mercurial itself, visit the Mercurial homepage at https://www.mercurial-scm.org/.

How to Install Mercurial

Install Mercurial client software using the JDeveloper Check for Updates feature, and then run Mercurial commands from the Mercurial option available in the Team menu. Use the procedure to install Mercurial.

Before you can take advantage of the Mercurial support, you need to have Mercurial client software installed on your system. JDeveloper's Mercurial support works by using the same commands as the Mercurial command line interface.

Mercurial is available through JDeveloper's Check for Updates feature. After you set up Mercurial, you can run Mercurial commands from the **Team** > **Mercurial** menu at the top of the JDeveloper main window.

To install Mercurial:

1. Select **Help** from the main JDeveloper window, then **Check for Updates**.
2. On the first page of the **Select Update Source** wizard, make sure **Search Update Centers** is selected.

3. Select **Official Oracle Extensions and Updates**.
4. Click **Next**.

   The wizard updates with a list of extensions you can install.
5. On the **Select Updates to Install** page of the wizard, select the checkbox next to **Mercurial VCS Extension**.
6. Click **Next**, then **Finish**.

   The Mercurial extension installs. Installation is complete after you restart JDeveloper.

You can also access the Check for Updates wizard by selecting **Tools** > **Features** > **Check for Updates**. For more information, see Working with Extensions.

After the Mercurial client is set up, you can run Mercurial commands from the main window by selecting **Team** > **Mercurial**.

How to Set the Path to the Mercurial Executable

If the `hg.exe` file for Mercurial is not on the system path, then set the path to the file. Use the procedure to set the path to the Mercurial executable file.

After installing through Check for Updates, you may set the path to the hg.exe executable file. hg.exe is the executable file for Mercurial. You only need to set the executable file if is not on the system path.

To set the path to the Mercurial executable file:

1. On the main JDeveloper window, select **Tools** >**Preferences**.
2. In the left pane of the Preferences dialog, expand **Versioning**, then click on **Mercurial**.
3. On the **Versioning: Mercurial** page you can select:
   - The name of the executable that will be used to run Mercurial, for example, hg.exe.
   - One of the installations of hg.exe in the system or different revisions of it, if they exist.
   - Some other location of hg.exe if, for example, it isn't on the system path.

How to Clone an External Mercurial Repository

Cloning a Mercurial repository creates a copy of the entire repository to work with in JDeveloper. Use the procedure to clone a Mercurial repository.

A repository is a directory that contains source files, along with their complete histories. Cloning makes a complete copy of another repository so that you have a local, private version of it to work with. You can create a local repository in any directory where you have write permission.

When cloning, you effectively create a copy or clone of the entire repository to work with in the IDE. To do so, you need to be able to access a Mercurial repository that you have read privileges for.

To clone a Mercurial repository:

1. On the main JDeveloper window, select **Team** > **Mercurial** > **Clone**.
2. On the Clone Repository window, enter the **Source Location** of the repository.

   This is the location of the repository that is to be cloned. You can enter either a URL or local path, for example, `http://selenic.com/hg`.

3. In the Destination field, enter a destination for the local repository, for example, `C:/JDeveloper/mywork/hg1`.
4. Enter a **User Name** and **Password** for the remote repository, if required.
5. Click **OK**.

How to Place Projects Under Version Control

Placing a project under version control creates a new local Mercurial repository in the current directory and imports sources into it. Use the procedure to place a project under version control.

You can place any project you are working on under version control. This creates a new local Mercurial repository in the current directory and imports your sources into it. The repository files are placed under a `.hg` directory under the project directory.

To place a project under version control:

1. In the Projects window (located on the left side of JDeveloper), select an unversioned project.
2. From the JDeveloper main window, select **Team** > **Mercurial** > **Initialize**.

   You can view files being added to the repository and their status from the Messages Log window.

   Once complete, all the project files are registered in the repository as Locally New.
3. Select **Mercurial** > **Commit** from the project's context menu to commit these project files to the Mercurial repository.
4. Enter a message about the change being committed in the **Commit Message** text area, and then click **Commit**.

   The committed files are placed together with the `.hg` directory in the Mercurial repository directory.

How to Merge File Revisions

When you merge changes between repository revisions and the local working copy, the working copy is updated with all the changes that were made. Use the procedure to merge file revisions.

You can merge changes between repository revisions and your local working copy. The current working directory is updated with all changes made to the requested revision since the last revision.

To merge file revisions:

1. From the JDeveloper main window, select **Team** > **Mercurial** > **Initialize**.
2. In the **Working Directory** field of the Merge Working Directory dialog, enter the top-level directory in the repository, for example, `c:\JDeveloper\mywork\hg1`.
3. Check the **Use Revision** checkbox and enter the revision number.

   If you don't know the number for the revision, click **Select Revision**. A dialog displays the available revisions number listed from most recent, the revision date, the user who made the revision, and comments.

How to Commit Changes

If working copies of version-controlled files have been edited, the Mercurial Commit action places the changes in the repository. Use the procedure to commit changes in local files to the repository.

Once your working copies of version-controlled files have been edited, you can then place changes into the repository using the Mercurial Commit action.

It's a good idea to update any copies you have against the repository prior to performing a commit in order to ensure that conflicts do not arise. This updates the local repository to include the latest changes.

To perform an update on sources that you have modified:

- In the JDeveloper main window, select **Team** > **Mercurial** > **Update**.

To commit changes in local files to the repository:

1. Select a version-controlled file (for example, from the Projects window) and right-click.
2. Select **Versioning** > **Commit** from the context menu.

   The Commit Dialog opens, listing all files that contain local changes.
3. Enter a commit message in the Commit Message text area, indicating the purpose of the commit.

   You can also select either a comment template you created in **Tools** > **Preferences** > **Versioning** > **Templates** or one of the comments that you have previously entered in the Commit dialog.
4. Click **OK**.

   JDeveloper executes the commit and sends your local changes to the repository. You can view files being committed to the repository in the Messages log.

   Use **Commit All** to view all the changed files that you can commit to the Mercurial repository. You can commit selected files or all outstanding changes. Use the `Shift` or `Ctrl` keys to select which files you want to commit.

Setting Up and Configuring a Source Repository

Configure the source repository after you have initialized the version control system in JDeveloper. Load the repository with content before you use Subversion with JDeveloper.

After initializing your version control system in JDeveloper, the next important step is to configure the source repository. Typically, you maintain a local repository, containing local copies of the files you are working with, on your own system. You typically check out the files you want to work on, make edits to the versions on your local system, and then check the files back in to the remote repository. Your version control system typically tracks, or at least notifies you of, changes and conflicts if more than one person is editing a file at a time. The menu options and details vary from system to system; these variations are described individually in each of the following sections.

Before beginning to use Subversion with JDeveloper, you will need to load your repository with content so that you have local versions to edit. For more information, see How to Load the Repository with Content.

How to Create a Source Repository

For every project or release, a repository has to be created. The source repository is used to check out and check in files.

Creating a repository is something you typically only do once per project/release; once you have created the repository, you check files in and out as part of your daily work routine. In many teams, the source repository is created by a team member assigned to the role of administrator for the repository; if this is the case, you may be able to skip this section and rely instead on accessing the existing source repository for checking out and checking in files.

The details of creating and connecting to your source repository differ depending on the versioning software your team uses. The following sections include instructions on creating source repositories for the versioning systems available to JDeveloper.

Creating a Subversion Repository

A Subversion repository may need to be created on the local file system through JDeveloper which will access the repository using the `file:///` protocol. Use the procedure 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 **Team > 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 Versions window.
2. Double-click on a folder to view its contents.
3. Right-click on an element to view available operations.

Initializing a New Git Repository

A new project with all new files requires a Git repository initialization. Initialization is done once for a project, at the beginning of the project. Use the procedure to initialize a Git repository.

If you have new files that are not already part of any existing Git repository, you need to initialize a Git repository.

You initialize a Git repository when you have a new project with all new files. This is typically done once per project, at the beginning. As an ongoing task, you are more likely to add new files to an existing Git repository (see Adding New Files to an Existing Git Repository), or check out files, edit them, and then commit the changes back to the Git repository (see Committing a Change to the Git Repository).

To initialize a Git repository:

1. Select **Team > Git > Initialize**. This displays the Initialize Repository dialog.
2. Enter the path to the local repository, then click **OK**.

Making a Local Copy of an Existing Git Repository

Use Git, HTTP, HTTPS, or Secure Shell protocol to make a local copy of a central Git repository. Use the step to clone a Git repository.

If your team already has a central Git repository and you wish to make a local copy of it for your work, you can clone the Git repository as described here.

The following protocols can be used to clone Git repositories:

- Git (`git://`). The simplest way of connecting to the Git server. The connection is not authenticated.
- HTTP (`http://`). HTTP is another simple way of connecting to the server. http:// connections are insecure even with authentication as the files will be transferred in clear.
- HTTPS (`https://`). Connects to the server using password authentication. https:// connections are fully secure and will work through a web proxy.
- Secure Shell (`ssh://`). This uses public key authentication, and you will need to generate an authentication key using a utility such as `ssh-keygen`. This is the most secure way of cloning an existing Git Repository. However, it will require additional setup if you are trying to connect through a web proxy.

To clone a Git repository:

- Select **Team > Git > Clone**.

This opens the Clone from Git Wizard. Enter the information requested on each screen. You can press **F1** or click **Help** at any time for more information.

Adding New Files to an Existing Git Repository

To add new files to an existing Git repository, select, add the files, and then commit them. Use the procedure to add new files to an existing Git repository.

Adding new files to an existing Git repository involves selecting, then adding the files, and finally committing them for the changes to be incorporated into the repository.

To add new files to an existing Git repository:

1. Select the files to be added.
2. Select **Team > Git > Add** (or **Team > Git > Add All**). This displays the Add dialog.
3. Click **OK**.

Creating a Local CVS Repository

If external CVS client software is in use, you can create a new CVS repository on a local file system. Use the procedure 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 **Team > CVS > 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.

Importing JDeveloper Project Files Into CVS

To use a JDeveloper project with CVS, import the project files into the CVS repository so that they are placed under source control. Use the procedure to use the Import to CVS wizard.

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 **Team > CVS > 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.

Bringing Files Under Perforce Control

Files to be brought under source control are placed in the Perforce client workspace, and then submitted to a central location called the Perforce depot to version them and make them

accessible to users. Use the procedures to place JDeveloper files and files created outside JDeveloper under Perforce control.

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.

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 Applications window and choose **Team > Perforce > 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 Applications window 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 Applications window and choose **Team > Perforce > 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 Applications window 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. Refresh the application or project.

   The files should now appear in the Applications window, 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.

How to Connect to a Source Control Repository

Use the Connect To... option in the Team menu to connect to a source control repository that has been created.

Once the source control repository has been created (either by you or by a team administrator), you typically connect to it from the **Team** menu, then selecting your versioning system from the **Connect To...** option.

Viewing Subversion Repository Content

The current content of the Subversion repository can be viewed by means of the Versions window. The folders offer the New, New Remote Directory, Delete, and Check Out options.

You can view the current content of the Subversion repository through the Versions window. 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 Versions window, 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.

Logging In to CVS

Independent of making connections, some types of connection to a CVS repository require logins. Use the procedure to log in to a CVS repository.

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 **Team > 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**.

Accessing Local Files with CVS

The path that JDeveloper uses to a CVS client on a computer will set the JDeveloper CVS preferences, and to use that CVS client by default, and if no path is found, preferences will need to be set to use the internal CVS client. Use the given step to access CVS through a firewall.

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 or higher for Windows platforms - `http://march-hare.com/cvspro/`
- 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
- 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.

Handling CVS File Types

For the automatic handling of binary files produced by JDeveloper, the CVS repository has to be configured by a CVS administrator.

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.

Connecting to Perforce

Connect Perforce to JDeveloper so that its operations become available within JDeveloper. Use the procedure to connect to Perforce manually.

Before Perforce operations become available within JDeveloper, you must connect to it.

To connect to Perforce manually:

1. Choose **Team > 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**.

Making Multiple Connections to Perforce

Multiple connections can be made to Perforce for work such as development and testing. Use the procedure to create a named Perforce connection.

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 **Team > 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. For more information on changelists, see How to Use Change Sets and Changelists.

Configuring JDeveloper for the Source Repository

Subversion requires no JDeveloper configuration for the source repository. Other version control systems have configuration requirements.

If you are using Subversion, which is included by default in JDeveloper, you do not need to configure JDeveloper for your source repository. You only need to connect to the repository, then you update your local working copy, check files out to work on them, and then check in your changed files on completion. Other version control systems, however, have configuration requirements, which require you to configure them, and JDeveloper, before use.

How to Configure CVS For Use with JDeveloper

Set up JDeveloper to use CVS, create a local CVS repository for storing files, configure a secure shell for communicating with CVS, select a character set, and log into CVS.

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.

JDeveloper supports SSH Levels 1 and 2 as access methods for CVS repositories.

Configuring for SSH Level 1 (SSH)

Configure SSH Level 1 so that it can be used for remote shell access. Use the procedure to configure SSH Level 1 to enable remote shell access.

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.

Configuring for SSH Level 2 (SSH2)

Use SSH2 as an access method for the CVS repository in JDeveloper. Use the procedure to use SSH2 for remote shell access.

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 **Team > CVS > Administration > Generate SSH2 Key Pair**. If you are using an external CVS client, this menu option is unavailable.

Editing and Watching Files in CVS

When an external CVS client executable is used, both editing and watching become available. Use the procedures to set up JDeveloper to use editing and watching, and release an editor on a file.

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 Applications window, select **Team > 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 Applications window, select **Team > 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 Applications window, select a project containing files about which you want to be notified.
2. Select **Team > 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 Applications window, select the project containing the files about which you want to be notified.
2. Select **Team > 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 **Team > 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 **Team > 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.

How to Load the Repository with Content

Load the repository with content for the team to use the repository for the selected version control system.

Before you can use the repository for your selected version control system, you typically have to load the repository with the content your team will be working on. You might need to do this when:

- Your team begins to work on a new version of the project (especially if you are working on more than one version concurrently, such as a patch set and a major update)
- Your team is starting an all-new project, either by beginning with an older version or with file templates
- You are performing a clean installation of JDeveloper on a new workstation that does not yet contain a local file system to store your repository's files as you work on them

Normally, loading the repository with content is done once per project. After this initial load, you will regularly update the files and folders in your local repository to ensure that your files are up to date with the work of your team.

Importing JDeveloper Files Into Subversion

Import files into the Subversion repository from JDeveloper, and then check them out. Use the procedure to import an existing JDeveloper project or application 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 Applications window, select the application or project that you want to import into Subversion.
2. Select **Team > 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 Applications window 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.

Importing a Project to Subversion

Import an entire project into Subversion using the JDeveloper Version Application feature. Use the procedure to import files using Version Project.

You can also import an entire project into Subversion using the JDeveloper Version Application feature.

To import files using Version Project:

1. Select the application you wish to add to version control.
2. Select **Team > Version Project**. This opens the Import to Subversion wizard.
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 Applications window 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.

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.

Adding a File to Subversion Automatically

A new file in JDeveloper must be added, and then committed to Subversion control so that it can be used with the JDeveloper Subversion facilities. Use the procedure to add new files on commit.

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 > Subversion > General**.
2. Select **Automatically Add New Files On Committing Working Copy.**
3. Click **OK**.

Adding Files Individually to Subversion

Files can individually be added to Subversion. Use the procedure to place individual files under Subversion control.

You can also place individual files under Subversion control.

To place individual files under Subversion control:

1. Select the files in the Applications window and choose **Team > 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 Applications window 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 Applications window and choose **Team > 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 Applications window with an orange dot, indicating that they are known to the Subversion repository and are up to date.

Moving Files from Remote Repositories in Git

When working with remote Git repositories, in addition to `pull` and `clone`, it may be necessary to Git `fetch` content from a remote repository to bring changes into a local repository without changing any existing branches. Use the procedure to copy from a remote repository without changing existing branches.

While you use your local Git repository for changes made while editing, it is frequently necessary in a distributed team to work with remote repositories. Push, pull and clone are three concepts that apply to remote Git repositories.

Fetching and pulling are two different methods of obtaining content from a remote repository. When you fetch from a remote repository, Git loads all changes into your local repository but it does not change any of your existing branches. This way, you can inspect the changes and merge them as appropriate.

To copy from a remote repository without changing existing branches:

1. Select Team > Git > Fetch
2. Follow the instructions on the wizard screens to specify the remote repository and remote branch you wish to fetch.
3. Click Finish to fetch the files from the remote repository.

Pulling, on the other hand, copies files from the remote repository and updates the HEAD branch of your local repository.

To copy changes from your local repository to the remote repository, use the Push command.

Importing JDeveloper Files Into Perforce

To be placed under formal source control, files must be stored in the Perforce client workspace, and then placed in the Perforce depot so that they can be versioned and accessed by other users. Use the procedure to use the Import to Perforce wizard.

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.

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 **Team > Connect to Perforce**.
2. In the Applications window, select the JDeveloper project that you want to bring under Perforce control.
3. Choose **Team > 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 Applications window. 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.

Updating a Project, Folder, or File in CVS

Update individual files or the contents of a project folder using CVS update operations, or replace local files with files from the CVS repository. Use the procedures to update an individual file, to update the contents of a project folder, and to update files shown in the Pending Changes window.

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 Applications window, and then choose **Team > 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 Applications window 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.

> **Note:**
>
> For the Pending Changes window to be populated with candidates, the project containing those candidates must be open. If you do not see a file that you expect to see, open the project (**File > Open > Project**, then select your project).

How to Create a WebDAV Connection

Web-based Distributed Authoring and Versioning is an extension to HTTP which allows users to collaborate in editing and managing files on WebDAV-enabled servers. .JDeveloper acts as a WebDAV client to connect directly to any Oracle Internet File System allowing users to view WebDAV files from the database.

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.

WebDAV Server Requirements

Run a WebDAV server, Apache 1.3.19 or above, and conforming to the WebDAV 1.0 standard to use JDeveloper as a WebDAV client.

You must run a WebDAV server to use JDeveloper as a WebDAV client. The WebDAV server must be one of the following:

- 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`

Creating a WebDAV Connection

A WebDAV connection in JDeveloper makes it possible to view files and folders as part of a JDeveloper project. Use the procedure to create a WebDAV connection in JDeveloper.

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**.

Accessing a WebDAV-Enabled Server Via a Proxy Server

Configure JDeveloper before accessing WebDAV-enabled servers on the internet if a proxy server is used to access the internet. Use the procedure to access a WebDAV-enabled server by means of 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.

Modifying a WebDAV Connection

WebDAV connections that are shown in the Application Resources section of the Applications window can be modified. Use the procedure to modify a WebDAV connection.

WebDAV connections are shown in the Application Resources section of the Applications window, 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**.

Refreshing a WebDAV Connection

WebDAV connections that are shown in the Application Resources section of the Applications window can be refreshed to reflect the current contents of the WebDAV server. Use the procedure to refresh the contents of a WebDAV connection.

WebDAV connections are shown in the Application Resources section of the Applications window, 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**.

Deleting a WebDAV Connection

Delete a WebDAV connection from JDeveloper without the deletion affecting any file or folder on the WebDAV server. Use the procedure to delete a WebDAV connection.

WebDAV connections are shown in the Application Resources section of the Applications window, 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.

Working with Files in Source Control

The workflow in a version control system involves checking files out, modifying them, and checking them in. Files are locked to prevent file conflicts.

As a general rule, your workflow in the version control system your team uses will follow this basic format:

- Check out files from the repository
- Make changes
- Check in (or commit) files back to the repository

In some circumstances, you may find that other team members have been editing the same files that you are committing. This requires resolving file conflicts.

In addition, many version control systems allow you to lock a file so that other users cannot check it out. This prevents you from having file conflicts which need to be resolved.

How to Check Out Files

A file needs to be check out from the repository before it is changed. A check out normally locks the file and prevents file conflicts.

Commonly, your version control system requires you to check out a file from the repository before you begin making changes to it. This logs your access to the file and, in many

instances, locks the file you have checked out to prevent other team members from accessing it while you are editing it. This can help prevent problems where multiple team members make conflicting edits to the same file.

Checking Out Files from the Subversion Repository

Subversion files that are checked out from the repository are copied on the local computer constituting the Subversion working copy. Use the procedure to check out modules from the Subversion repository.

To begin making edits and revisions to files in your project, 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 up to date.

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 Versions window, select the repository node or folder containing the files that you want to check out.
2. Choose **Team > Subversion > 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**.

Checking Out Files in Git

Check out a specific version of a file from the Git repository to modify it. Use the procedure to check out a file.

Checking out a file from your Git repository makes that file available for changes and edits. You can also specify the revision you wish to check out.

To check out a file:

1. Select **Team > Git > Checkout**. This displays the Git Checkout Revision dialog.
2. Enter the branch from which you wish to check out the file. To browse from a list of available branches, click the **Branch** button.
3. Select the tag (optional) you wish to check out. To view a list of tags, click the **Tag** button, then select the desired tag.
4. Specify the commit ID (optional) for the checkout. To view a list of commit IDs used in your repository, click the Select Commit button, then choose from the list of commit revisions.
5. Optionally, create a new branch for the checkout.
6. When you have made all your selections, click **OK**.

Git checks out the files you have selected. They are now available for editing.

Checking Out CVS Modules

After importing a JDeveloper project into the CVS repository, you can check out modules from the CVS repository. Use the procedure to check out modules from the CVS repository.

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 by choosing **Team > CVS > Check Out Module**.

   Alternatively, you can select **Check Out Module** from the contextual menu.

   The Check Out from CVS dialog is displayed.
2. Complete the dialog. For help when using this dialog, press F1 or click **Help**.

Editing Files in Perforce

Use the Open for Edit option available in Perforce for the Team menu to place a file on a changelist thereby locking the file for editing.

Unlike some version control systems, Perforce does not require you to explicitly check out a file. To begin editing a file in Perforce, select **Team > Perforce > Open for Edit**. This opens the Open Files for Edit dialog, which gives you the option of placing the file on a changelist, locking the file so that other team members cannot edit it simultaneously, and more.

You can start editing a file under Perforce control just by opening it from the Applications window. 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.

How to Update Files with Subversion

Updating files at the beginning of every work session is good practice to ensure that files have all the changes that were made and were checked in by team members.

Updating files at the beginning of every work session helps ensure that you have all the changes made and checked in by your team members. This reduces the amount of time you might have to spend reconciling changes later, because it helps ensure that you have the latest version of your files when you begin editing them.

Updating Files from the Subversion Repository

Update local copies of files with files from the repository using the Update Working Copy option so that all the files in a checked-out working copy are updated. Use the procedure to update a working copy.

Once your Subversion repository is set up, you typically update your local working copy with files from the repository. This ensures that the files you work on contain all committed changes that others on your team may have made to the same files.

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 Applications window. 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 Applications window.

To update a working copy (recommended):

1. In the Applications window, select a navigator node or file that is part of the working copy.
2. Select **Team > 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**.

Updating Individual Files in the Subversion Repository

Update individual files, although not all files that have been modified by team members may get updated. Use the procedure to update individual files.

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 Applications window, select the file(s) that you wish to update and choose **Team > 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**.

Removing Files from Subversion Control

Use the JDeveloper Delete feature to remove a file from Subversion control. Use the procedure to remove a file 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 Applications window, 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**.

Working with Files in CVS

Working with files in CVS involves checking out the latest version of a file, editing it, and checking the file in with changes. In CVS you can refresh the display of CS objects, add and remove files, compare files, or lock and unlock files.

As a 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.

Refreshing the Display of CVS Objects

The Applications window displays source control status of objects by means of icons.

The source control status of an object is indicated in the Applications window 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.

Table 6-1 (Cont.) CVS Object Status

Icon	Description
	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 sync 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.

Refreshing the Status of Objects in JDeveloper

A manual refresh of an object ensures that the status indicated in the Applications window matches the true status of the object in the source control system. Use the step to refresh the status of objects in JDeveloper.

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 Applications window 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 Applications window or CVS Navigator, click **Refresh**.

Adding and Removing Files in CVS

A file that is part of a project that is under CVS version control can be added to CVS. Use the procedures to add a file to CVS through the Applications window, to add files shown in the Pending Changes window, and to remove a file from CVS.

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 Applications window by the icon **+**.

After completing setup, your work will revolve around the following:

- Updating your files from the repository
- Checking out the files you need to work on
- Editing them in JDeveloper

- Committing the modified files back to the repository

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 CVS control as the project changes. Finally, you might use special properties of the files associated with specific versions for tracking bugs, customer requests, and other characteristics.

To add a file to CVS through the Applications window:

1. Select the file in the Applications window and choose **Team > Add** (or, if the file is binary, **Team > Add as Binary**). JDeveloper usually recognizes binary files and adds (Binary) after the file name in the Applications window. 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 Applications window, select one or more files to be removed, then choose **Team > 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.

How to Work with New and Changed Files in Git

In the Git version control system, files are committed to import new files to a repository, and to check in changes after a file edit.

In the Git version control system, you commit files for two reasons: to make your initial import of new files to your repository, and to check in changes when you have made edits.

In Git, the basic file workflow is in three parts:

1. You initially create a local file system with the selected branch and content from your repository. You can do this using **Team > Git > Clone**.
2. You add new files to the repository: one at a time using **Team > Git > Add**, or multiple files with **Team > Git > Add All**.
3. As you edit files, you check them back into the Git repository one at a time with **Team > Git > Commit**, or multiple files with **Team > Git > Commit All**.

The following sections describe how to perform these operations.

Adding a File to a Git Repository

The Git menu provides the Add option to add single files to the Git repository. Use the procedure to add a file.

To add a single file to your Git repository, use the Add command from the Git menu.

To add a file:

1. Select **Team > Git > Add**.
2. Browse to the pathname for the file you wish to add.
3. Click **OK**.

The file is added to your selected Git repository.

Adding Multiple Files to a Git Repository

The Git menu provides the Add All option to add multiple files to a Git repository. Use the procedure to add multiple files.

To add multiple files to your Git repository, use the Add All command from the Git menu. This command adds all files in a directory.

To add multiple file:

1. Select **Team > Git > Add All**.
2. Browse to the pathname for the directory from which you wish to add files.
3. Click **OK**.

All files in the selected directory are added to your Git repository.

Create a Git Stash

Use Stashing to store modified tracked files and staged changes outside the working directory so that they can be reapplied later. Use the procedure to create a stash.

Stashing becomes handy when you want to switch branches and don’t want to loose your unfinished work. It helps in keeping modified tracked files and staged changes from your working directory and stores in an area outside of the working directory that you can reapply at any point of time later. You can save as many stashes as you want and can apply them later.

To create a stash:

- Select **Team > Git > Stash Changes**.
- Select the file(s) from the list that you wish to include in the stash and type in some definitive comment in the Comments area.
- Click **OK**.

The stash is added to your selected Git repository under the Stashed Commits section.

Committing a Change to the Git Repository

A file that has been changed has to be committed to the Git repository. Use the procedure to commit a change.

When you have made changes to a file, you can commit them to the Git repository.

To commit a change:

1. In the Applications window, select the file whose changes you want to commit.
2. Select **Team > Git > Commit**. This displays the Commit dialog, which displays the file name and location in your local directory.
3. In the Commit dialog, you can optionally choose to commit non-staged files.
4. In the Comments field, enter a short description of the changes you have made to the file you are committing.
5. If your team uses comment templates, select one from the Comment Templates drop-down list. You can also select the link to display the **Preferences > Git > Comment Templates** dialog. This allows you to add or import comment templates.
6. When you are finished, click **OK**.

JDeveloper commits your changed file to the Git repository.

Committing Multiple Files to the Git Repository

Changes to multiple files can be collectively committed to the Git repository. Use the procedure to commit multiple changed files.

When you have made changes to multiple files, you can commit them all to the Git repository in one operation.

To commit a number of changed files:

1. In the Applications window, select the files whose changes you want to commit.
2. Select **Team > Git > Commit All**. This displays the Commit dialog, which displays the path name to the files in your local directory.
3. In the Commit dialog, you can optionally choose to commit non-staged files.
4. In the Comments field, enter a short description of the changes you have made to the file you are committing.
5. If your team uses comment templates, select one from the Comment Templates drop-down list. You can also select the link to display the **Preferences > Git > Comment Templates** dialog. This allows you to add or import comment templates.
6. When you are finished, click **OK**.

JDeveloper commits your changed files to the Git repository.

Applying a Git Stash

Apply stashes that have been created to the original files in the working directory. Use the procedure to apply a stash.

When you have created one or more stashes, you can apply them later to the original files in working directory. To apply a stash:

1. Select **Team > Versions**. This displays the Versions navigator in the left panel which displays your stashed commits under Git repository tree structure.
2. Right click the stash that you with to apply and select **Apply Stash**. This opens the Apply Stash dialog.
3. Click **OK**. Optionally you can select the **Delete stash after applying** checkbox before clicking **OK** if you do not need the stash anymore.

The selected stash is applied to your original file(s).

How to Work with Files in Perforce

Perforce provides methods to determine changes between two revisions of a file, and to apply the changes to another file. Perforce also supports the export of details about repository connections and files in the repository.

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.

Synchronizing Local Files With the Controlled Versions in Perforce

View the latest file statuses, and update files to synch with the repository version. Use the procedure to update files to match the controlled file 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 Applications window and choose **Team > 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.

Synchronizing Files With the Perforce Navigator

Use Perforce Navigator to update a working directory with content from the depot. Use the procedure to synchronize files using the Perforce Navigator.

The Perforce Navigator lets you browse the Perforce depot and update your working directory from content at the depot. Using the Applications window, 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 open 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 Versions window, 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.

Filtering Files By Perforce Workspace

To simplify working on several files, set filters in the Perforce client so that the filtered file view is displayed in JDeveloper. Use the step to filter files in JDeveloper.

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 Applications window would be filtered so only `//depot/JDeveloper` is shown.

Refreshing the Status of Files under Perforce Control

Icons display the source control status of a file in the JDeveloper navigators. Use the step to refresh the status of files in JDeveloper.

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.
	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 Applications window 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**.

Deleting Files from Perforce

Delete a file that it is under Perforce control by using the Perforce features within JDeveloper or the Perforce client application. Use the procedure to delete a file under Perforce control.

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.

If you need to retrieve a file that has been deleted, you will need to use the Perforce client. To do this, select **Team > Perforce > Launch Perforce Client**.

To delete a file under Perforce control:

1. Select the file in the Applications window and choose **Team > 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 Applications window.

How to Lock and Unlock Files

In some version control systems, checking out a file automatically locks it, and checking it in unlocks it.

Not all version control systems explicitly require, or even have the facility for, locking files before editing. In some systems, checking out the file automatically locks it and prevents other team members from editing that file. In other systems, the files can be checked out by any team member, with the emphasis being on easily merging different changes to the resulting file.

The following sections explain how each version control system handles locking and unlocking a file.

Locking and Unlocking Files in CVS

A file that is locked in the CVS repository prevents other users from accessing it, and unlocking the file restores file access to other users. Use the procedures to lock files in CVS, and to unlock files in CVS.

> **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 Applications window, choose **Team > 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 Applications window, choose **Team > 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**.

Editing Files in Perforce

Open a file under Perforce control from the Applications window, or manually open files for editing by setting the Automatically Open Files for Edit preference to off. Use the procedure to edit a file under Perforce control.

By default, you can start editing a file under Perforce control just by opening it from the Applications window, without explicitly locking the file. 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 Applications window and choose **Team > 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 Applications window.

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.

Checking Out Files in Team System

Files under Team System source control must be checked out so that you can work on them. Use the procedure 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 **Team > Check Out.**

   The Check Out dialog is opened.
3. Complete the dialog.

   For information while using the dialog, press **F1**.

Viewing the Status of a File in Team System

The status of a file that is under Team System source control can be checked. Use the procedure to view the status of a file.

You can check the status of a file that is under Team System source control. See also Refreshing the Status of Files in Team System.

To view the status of a file:

1. With the file selected in the Applications window, open the context menu and select **Team > 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.

Refreshing the Status of Files in Team System

Icons display the source control status of a file in the JDeveloper navigators.

The source control status of a file is indicated in the JDeveloper navigators by icon overlays, as below.

Table 6-3 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**.

How to Check In Changed Files

When you check in a file, the local file version that has changed is uploaded to the central repository.

Checking in, sometimes called committing, is the process by which the changes you have made to your local file version are uploaded to the central repository. The following sections outline the procedures used by the various version control systems in use with JDeveloper.

Committing Files to the Subversion Repository

Edited and revised working files must be committed as individual files or as change sets to update the Subversion repository with the latest version of the files. Use the procedures to commit a working copy from the Applications window or from the Pending Changes window.

Once you have made edits and revisions to your working files, you return them 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 Applications window or the Pending Changes window:

- Select the file(s) and choose **Team > 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 Applications window:

1. Select a navigator node or file that is part of the working copy.
2. Select **Team > 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 committed, regardless of which node you have active in your application in the JDeveloper Applications window. 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 Applications window.

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 **Team > Commit Working Copy**.

Saving Work Item ID with the Oracle Team Productivity Center Extension

Committing a file automatically saves a work item ID as a tag in the comment dialog.

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.

Committing Changes to the Git Repository

You can either select an individual file to commit, or commit all files simultaneously. Use the procedures to commit an individual file, and to commit all files in a project.

There are two options for committing your changes to the Git repository: selecting an individual file to commit, or committing all files at once.

To commit an individual file:

1. Select the file you wish to commit.
2. Select **Team > Git > Commit**. This opens the Save Files dialog. Click **OK** to continue. This opens the Commit dialog.
3. If your team uses comment templates (for example, to make it simpler to track issues automatically in a bug-tracking system), select the template or add a comment. Otherwise, click **OK** to commit the file to the repository.

To commit all files in a project:

1. Select the project whose files you wish to commit.
2. Select **Team > Git > Commit All**. This opens the Save Files dialog. Click **OK** to continue. This opens the Commit All dialog.
3. If your team uses comment templates (for example, to make it simpler to track issues automatically in a bug-tracking system), select the template or add a comment. Otherwise, click **OK** to commit the files to the repository.

Committing Changes to CVS

Update the CVS repository with the latest version of working files, and add, or remove files from the repository. Use the procedures to commit individual files or the contents of project folders shown in the Applications window, and to commit files shown in the Pending Changes window.

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 Applications window:

1. Select the file(s) in the Applications window, and then choose **Team > 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 Applications window:

1. Select the project folder(s) in the Applications window 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.

Submitting Changed Files to the Perforce Depot

Changes made to a file are initially saved in the Perforce client workspace until they are submitted, and made available to other users. Use the procedure to submit changes 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 Applications window, choose **Team > 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 Applications window with a green dot, indicating that it is up to date compared with the version in the Perforce depot.

How to Check In Files to Team System

A file that is checked in to the Team System server can be seen and worked on by other users. Use the procedure 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 Applications window, select the file that you want to check in.
2. Select **Team > Check In**.

   The Check In dialog is opened.
3. Complete the dialog.

   For information while using the dialog, press **F1**.

How to Use Change Sets and Changelists

Version control systems such as Subversion and Perforce use change sets or changelists to group files, which form part of a single logical group, together.

Some version control systems (notably Subversion and Perforce) use the notion of change sets or changelists to group together files which form part of a single logical group. For example, you might be adding a new feature to an application which has a number of specific files: the HTML framework which calls a JavaScript function, the JavaScript file which contains the function itself, and a set of image files which represent the various states of the button before, when and after the user clicks on it in the final application. You might choose to group all these files in a single change set, as a way of tracking them and ensuring that you commit all of them to the repository (or that you refer to all of them in your team's tracking or build software).

The following sections describe how to use change sets or changelists in the version control systems that support them.

Using Change Sets in Subversion

Adding files to change sets or change lists enables group operations on working copy files. Use the procedure to add a selected file to a new change set.

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 **Team > 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**.

Editing Change Sets

For every installed version control system, JDeveloper creates a default, unnamed change set. You can edit, delete, or set a selected change set as the default.

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.

Creating a Perforce Changelist

Use the procedure to create a Perforce changelist. In Perforce, grouping files into changelists simplifies working on the files.

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.

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**.

How to Annotate a Perforce Revision or Changelist

Annotations to a Perforce revision or changelist are saved as comments with links to every file in the revision making it possible to see changes as annotations to files. Use the procedure to add annotations to a 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 **Team > 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.

Adding Files to a Perforce Changelist

A Perforce changelist makes it possible to manage several changed files and folders through single operations. Use the procedure to add files to a 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.

Submitting a Perforce Changelist

If files have been added to a changelist, a series of edits to the files can be submitted in a single operation. Use the procedure to select and submit the files in a 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.

Using the Perforce Changelist Browser

The Changelist Browser displays the name, description, and contents of every pending changelist, and allows you to browse the files that are associated with a changelist. Use the procedures to create a changelist with the Changelist Browser, submit it, and to move files between changelists.

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**.

How to Use Comment Templates for Checkins

Configure the comment templates in Subversion, CVS, Git, and Perforce to check in or commit files. The comment text in comment templates can be structured in a way that can be read by a bug-tracking system to associate file checkins with the issues that are being tracked.

Subversion, CVS, Git, and Perforce all let you configure comment templates to be used when checking in or committing files. One popular use for a comment template is to structure the comment text in such a way that it can be read by a bug-tracking system to correlate file checkins with issues being tracked. For example, beginning a comment with the string `bugtraq` and then following with the issue number would make it possible to analyze comments automatically and generate a report of files and the bugs they were filed against.

The four version control systems that use comment templates all allow you to are Subversion, CVS, Git, and Perforce. The following instructions are common to all these systems.

Creating Templates

In JDeveloper, create and select templates from the Commit or Checkin menu that provides for file checkin comments, which can associate the edits with the issues being tracked. Use the procedure to create a new template.

Many team environments allow 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.

In particular, some teams wish to correlate checked-in files with the bugs that the files address. One way to do this is to set up a standard template for issue tracking, for example, `bugtraq <bug number>` where you replace *`<bug number>`* with the actual ID of the issue this file addresses, and use that as the comment to a file checkin. This way, file checkin comments can be used to correlate the edits with the issues being tracked.

JDeveloper lets you create and select templates for use with such comments. The templates are available from the Commit or Checkin menu, depending on how your version control system refers to the process of committing changes to the repository.

To create a new template:

1. Select **Tools > Preferences > Versioning > Subversion > Comment Templates**.
2. Click **Add**. This brings up the Add Template dialog.
3. Type in the comment template as you wish to use it, then click **OK**.

The template you created will now be available to select when you commit or check in your files.

Sharing Templates

Import and export templates as text files, and share a set of templates within a team. Use the procedures to import or export a template.

JDeveloper lets you import and export templates as text files, allowing you to share a set of templates with your team. This simplifies using the exact format that any tracking system you use will expect to see in the commit comments, and also avoids errors in formatting the template.

To import a template:

1. From the Comment Templates page, click **Import**. This opens a file browser.
2. Navigate to the directory containing the template file you wish to import and click on the file to select it.
3. Click **Open**.

The template file will be imported to JDeveloper.

You can also export your templates as a file to be imported later, either by you or (if you make the file available on a shared resource) by your team members.

To export a template:

1. From the Comment Templates page of your selected version control system, click **Export**. This opens a file browser.
2. Navigate to the directory in which you wish to save your template file. Note that the default template file format is XML.
3. Give your template file a name (for example, `my_bugtraq.xml`) and click **Save**.

The template file is now available for import to JDeveloper.

Note that if your team uses more than one version control system, you can use Export and Import to share the same templates across systems.

Selecting and Using Templates at Checkin

Using the templates that are available from the Commit menu, enter comments when a file is checked in, and associate the edits that were made with the issues that are being tracked. Use the procedure to select a template.

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 select a template:

1. From the Commit menu for your selected versioning system, click **Choose a template or previous comment**.
2. Select the template from the list.
3. If your template requires you to enter specific data (for example, a bug or enhancement tracking number), make the edits in the Comments field.
4. Click **OK**.

Your file will be committed with the template you selected, plus any edits you made.

Shelving and Unshelving Team System Files

Shelving saves file changes in the Team System server without file check-ins thus allowing you to work on the changed files, to remove them from view, and revert to unchanged versions. Use the procedures to shelve and unshelve a set of file changes, and to delete a shelveset.

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 Applications window, select the versioned project containing the files.
2. Select **Team > 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 Applications window will change to reflect the new file statuses, if any.

To unshelve a set of file changes:

1. In the Applications window, select the versioned project into which you want to unshelve the file changes.
2. Select **Team > 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 Applications window will change to reflect the new file statuses.

To delete a shelveset:

1. Select **Team > 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**.

Deleting Team System Files

Delete a file from a workspace and from the Team System server. Use the procedure to delete a file.

Use to delete files from your workspace and from the Team System server.

To delete a file:

1. Select the file in the Applications window and choose **Team > Delete**.

   The Delete dialog is displayed with the file listed.

2. Click **OK**.

   On the Outgoing tab of the Pending Changes window (**Team > 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 **Team > 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.

Working with Branches and Tags

When working on multiple releases, use branches in Subversion, Git, and CVS to track, modify, and update each project for team work. Tags track file collections to capture the state of development at a specific point.

Many version control systems used with JDeveloper (notably, Subversion, Git, and CVS) use branches to keep track of development projects or streams. Branches are a useful way to separate out development when (as is often the case) your team is working on multiple releases -- for example, a patch release, a major update, and a new project -- at the same time. Tracking each of these projects as a separate branch in your version control system makes it easy to track, modify, and update each project with the correct feature set, bug fixes, and other changes, and to ensure that all members of the team are tracking the same issues on the same branch.

Tags, on the other hand, are a way of tracking collections of files that capture the state of development at a particular point. Where a branch represents a stream of development moving towards a particular release, product or version, tags are useful for tracking elements within a branch -- such as a specific bug fix that might affect multiple files in a given release.

How to Create Branches

In a version control system, a trunk represents the main line of development from which branches are created so that developers can work on files independent of the trunk.

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. Creating branches varies slightly from system to system.

Working with Branches and Tags in Subversion

In Subversion, merge a branch with the trunk, compare the differences between two revisions, and apply the differences to the current working copy. Use the procedures to create a branch or tag, to use the merge facility, and to switch the working copy to another location in the repository.

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 Applications window, select a project or file that is in the line of development that you wish to branch or tag.
3. Select **Team >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 Applications window, select a project or file that is in the start revision (that is, the resource that is to be compared against).
2. Select **Team > 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 Applications window, select a project or file that is in the current working copy.
2. Select **Team > Subversion > Switch**.
3. Complete the Switch dialog.

   For help when completing the dialog, press F1 or click **Help**.

Creating a New Branch in Git

Create a named branch of the Git repository, and apply a tag to the branch. Use the procedure to create a new branch in Git.

You can create a new, named branch of your Git repository, and optionally apply a tag when you create it, from the Create Branch dialog.This creates the named branch for use with Git, and associates the selected tag with it if you choose.

To create a new branch in Git:

1. Select **Team > Git > Create Branch**.

2. Enter the name for your branch or tag -- for example, the version number of the release you are working on, or the code name for the current update.
3. Enter the branch you are using as the source for the branch you are creating. To select from available branches, click **Select Branch**. This places the branch name in the Branch field and greys out the Tag field.
4. To give your new branch a tag, click **Select Tag**, then select from one of the available tags displayed in the Select Tag dialog.
5. Optionally, use a specific commit ID for the source of the branch you are creating. You can either enter the commit ID by hand, by copy and paste (the commit ID is a 40-digit hexadecimal number), or click **Select Commit** to choose one from a list of available IDs.
6. When you are finished, click **OK**.

Creating a New Branch in CVS

Create a new branch to start work on a project based on an earlier code version, or create a new branch from an existing branch of the code trunk. Use the procedures to create a new branch, and create a new branch from an existing 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.

How to Use Branches

Manage and track projects under version control by creating branches in the repository. Check out branches at the beginning of a project, check out individual files for edits or reviews, delete branches, or merge branches.

Once you have created a branch, JDeveloper presents a number of methods of using it to manage and track the projects you have under version control. Typically, you check out branches at the beginning of a project involving that branch, then later you can check out the

individual files that you are editing or reviewing. You may also occasionally need to delete a branch (when a project is archived, for example), and you may also need to merge a branch.

Checking Out a Branch in Git

To access files on a branch of code in a repository, you must check out the branch. Use the procedure to check out a branch in Git.

Checking out a branch in Git gives you access to the files on that branch.

To check out a branch in Git:

1. Select **Team** > **Git** > **Checkout**.
2. The Name and Location fields list the available repositories from which you can select the branch to check out. Select the repository you wish to use.
3. Click on **Select Branch** to display the list of available branches in the Select Branch dialog. Expand the folders as required to locate the desired branch. Select the desired branch, then click **OK**.
4. If you wish to check out content associated with a specific tag, click **Select Tag**, then choose the desired tag from the Select Tag dialog. When you are finished, click **OK**.
5. To check out content from an existing commit, click on **Select Commit**, then select the desired commit from the Select Commit Revision dialog. This lists all available commits (including their 40-digit hex ID), and also displays the date and time of the commit, the ID of the team member who made the commit, and a message field if notes were added at the time of the commit. When you have selected the commit, click **OK**.
6. Give the checked-out branch a name to associate with this branch, then click **OK**.

JDeveloper displays the branch that you have checked out, with any projects and files associated with the selected branch. You can access these files from the Application window.

Merging a Branch in Git

Changes on a branch must be committed after the changes from the branch have been merged with the Head.

Note that after merging your changes from the branch into HEAD, you must still commit those changes. The changes will be available in the main repository after they have been pushed to the main repository

Using Branches in CVS

The Tag, Branch and Merge menus provide access to the branches that have been created to work away from the main branch of a project. Switch between branches, merge a branch with the trunk, check out or check in files on a branch, and commit changes.

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).

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.

Switching the Branch or Version

Switch the branch or version of a file in JDeveloper. Use the procedures to select a branch, version or date from the Versioning menu, and to select a branch, version or date from the context menu of the project.

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 **Team > 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**.

How to Choose a Branch while Updating

When you update file content, you can also branch using the associated tag. Use the procedure to select a tag and branch while updating content.

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**.

Choosing a Branch While Checking Out

Use the procedure to select a branch while checking out a CVS module.

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 Versions window 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.

How to Create Tags

Tags identify logically grouped content such as a subset of files within a branch. Use tags to manage files in the version control system.

Tags give you another way of identifying content that is logically grouped in some way, typically as a subset of files within a branch (for example, a list of files associated with a particular feature set, or with a particular bug fix). Once you create tags, they can be used in a number of ways, in particular for checking out files from the version control system.

Creating Tags in Git

Either create a new tag when you check out a branch in Git, or create a tag using the Create Tag dialog. Use the procedure to create a tag in Git.

You have the option of creating a new tag when you check out a branch in Git. However, you can also create a tag at any time by using the Create Tag dialog.

To create a tag in Git:

1. Select **Team** > **Git** > **Create Tag**. This opens the Create Tag dialog.
2. In the **Name** field, type a descriptive name for the tag you are creating (for example, Version 1.2.3 Beta Release).
3. In the **Comments** field, enter any descriptive notes that can help you identify this content in the future when you are using this tag.
4. In the Branch field, enter the branch name or click **Select Branch** to open the Select Branch dialog, then choose the branch on which you wish to create this tag. When you have made your selection from the Select Branch dialog, click **OK** to close the dialog and make you selection.
5. If you wish to base this tag on content currently associated with another tag, click **Select Tag** to open the Select Tag dialog.
6. Optionally, use a specific commit ID for the source of the tag you are creating. You can either enter the commit ID by hand (or more likely, by copy and paste as the commit ID is a 40-digit hexadecimal number), or click Select Commit to choose one from a list of available IDs.

7. When you have entered all the information for this tag, click **OK**.

Creating and Assigning CVS Tags

Create and assign tags to content in CVS. Use the procedures to assign CVS tags, to view CVS tags, and to reset CVS tags.

CVS allows you to create and assign tags to content.

To assign CVS tags:

This procedure will assign symbolic tags to the local CVS revisions of selected files.

1. In the Applications window, 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 **Team > 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 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 **Team > 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 Applications window, select the file or files whose tags you wish to reset.
2. Choose **Team > Tag > Reset Tag**s.

Deleting CVS Tags

Delete the tag from a file, or from all the files within a project or workspace. Use the procedure to delete CVS tags.

Deleting a tag does not delete the content associated with it, but if you are using tags to identify a specific project that has completed, you can delete the tag to simplify the display when you are choosing tagged content in future.

To delete CVS tags:

This procedure will delete symbolic tags from the local CVS revisions of selected files.

1. In the Applications window, 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 **Team > 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**.

How to Use Tags

In Subversion, creating a tag is like creating a branch.

Using Tags in Git

Use the Tag option from the Team menu to select a tag in Git, and apply it to a selected file.

You can select from an existing tag in Git and apply it to the file you have selected for the current transaction.

To select a tag in Git:

- Select **Team > Tag**. Scroll through the list to find the tag you wish to apply to the selected file.

If there are more than one tags that apply to the file, hold down the Ctrl key, then select each tag that you wish to use.

Using Tags in CVS

In CVS, use tags as identifiers for branches, files, folders, or projects that are to be managed as logical groups. In addition to context menus, tags are available from the Tags option on the CVS menu in the Team category.

Tags in CVS 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 **Team > CVS > Tags** menu. The availability of tags differs depending on the context of the operations you are performing on your content.

Adding a Tag to a Project

A project that has a tag identifier can be accessed by selecting the tag from any CVS menu. Use the procedure 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 **Team > 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.

Applying Tags While Updating a Project or File

In the Update from CVS dialog, you can select and apply a tag. Use the procedure to select an existing branch, version, or date from the Projects view.

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 **Team > Tag**.
2. Choose **Use Revision, Tag or Date**.
3. Click the tag chooser icon.
4. Choose a tag from the list that appears.

How to Delete a Tag

Deleting a tag removes it from the list of available tags, and from the resources to which it had been applied without removing the associated content. Use the procedure 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 **Team > 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.

How to Use Properties in Subversion

With Subversion, you can define and add properties to files, folders, and other resources in the Applications window.

Subversion lets you define and add properties to various levels of the elements in the Applications window: 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.

About Subversion Properties

Associate a specific subversion property with a newly added feature to see all the files associated with the feature. Subversion allows you to specify the Resource file, Property name, Value string, Value file, and Set property recursively properties.

As an example of using subversion properties, 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.

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.

Working with Subversion Properties

Use View Subversion Properties to see a list of all elements that use a selected Subversion property, and to compare the Subversion properties between different versions. Use the procedures to view a list of Subversion properties, and to add a new Subversion property.

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 Applications window.
2. Select **Team > 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 Applications window.

2. Select **Team > 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.

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-
INFhttps://myserver.myteam.com/svn/repos/public-html/WEB-INF@16 ExternalWebInf
```

Viewing File and Property Status

Check the content status and any associated property status of a file that is under Subversion source control, or refresh the status of a file. Use the procedure to view the status of a 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 Applications window, open the context menu and select **Team > 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.

Resolving Property Conflicts in Subversion

If there are versions of files or folders with conflicting properties, resolve them using the Resolve Conflicts option in the Subversion menu for Team. Use the procedure to resolve Subversion property conflicts.

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 Applications window, select the element under Subversion control that has a property conflict.
2. Click the right mouse button, and then select **Team > Subversion > Resolve Conflicts**.

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.

Working with File History, Status and Revisions

Version control enables you to view file history, and compare the differences between file revisions.

One of the most useful parts of working with version control is the ability it gives you to look at the history of a file and compare changes made to different revisions. In debugging, you can look through the file history to find a version just prior to the introduction of a defect, which can help you identify where the problem occurred. If you and another team member make conflicting changes to a file or a set of files, you can go back to versions of the file from before the changes were made.

File History

Subversion provides the History Viewer that displays the change history of a file in the repository during the life cycle of a project. Use the step to view the history of a file.

You can refer to the history of a file in your repository to review changes made to it over the life of the project.

To help you understand the sequence of changes made to a specific Subversion file, you can use the History Viewer and view the history of Subversion files.

To view the history of a file:

- With the file selected in the Applications window, choose **Team > Version History** from the context menu.

  For more information while using the History Viewer, press F1 or click **Help**.

Refreshing the Status of Files Under Subversion Control

Icons in the Applications window and the Teams window show the source control status of a file, and this status can be manually refreshed. Use the step to refresh the status of files in JDeveloper.

The source control status of a file is indicated in the JDeveloper navigators (Applications window and Teams window) 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 Applications window 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**.

Replacing a File with the Subversion Base Revision

In Subversion, you can replace a file with its base revision. Use the procedure 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 Applications window, 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**.

How to Undo or Revert Changes

Cancel changes that have not been checked in by undoing changes, or reverting to specific, previous versions of files.

Undo changes, revert to previous versions, selecting specific revisions: these are all commonly used phrases to describe the process of canceling changes you have decided not to check in to the repository.

Reverting Files to their Previous State in Subversion

The Revert command in Subversion undoes changes, changes file status, and halts file removals. Use the procedure to revert a file.

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 Applications window or Pending Changes window and choose **Team > 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**.

Reverting Changes to Files in Git

The changes that were made to a file that has been committed to a Git repository can be reverted. Use the procedure to revert changes.

You can revert changes made to a file that you have committed to a Git repository to return it to a previous state.

To revert changes:

1. Select **Team > Git > Revert**. This displays a list of files that you can revert to a previous version, with the path in your local file system at which the file resides.
2. Select the file you wish to revert, then click **OK**.

Working with Revisions and Tags in CVS

In CVS, select a specific revision from the CVS repository to work on. Use the procedure to open a CVS file revision.

CVS lets you select specific revisions from the CVS repository, if you find it necessary to begin or resume working with a previous version of the file you are currently working on.

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 Applications window, choose **Team > 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**.

How to Merge Changes from Different Files

Merge changes from different files to get a single version which contains edits from many members of the team.

Many version control systems offer ways to let you merge changes from different files, ending up with a single version which contains edits from multiple team members. To begin, you will need to compare different versions of the file, then follow the procedures for selecting and merging changes.

Comparing Files in Subversion

Files that are under Subversion control can be compared with other revisions of the same files, or with other files. Use the procedures to compare revisions of a file, to compare a file with another file, and to compare two files.

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 Applications window.
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.

Resolving Conflicts in File Versions

In the event of a conflict between the working copy of a file and the one in the Subversion repository, resolve the conflict or revert to a non-conflicting version of the file. Use the procedure to resolve the conflicts using the merge tool.

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 (**Team > 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 Applications window and choose **Team > Revert.**

To resolve the conflicts using the merge tool:

1. Select the file in the Applications window and choose **Team > 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 Applications window and choose **Team > Mark Resolved**.

Using the Merge Tool to Resolve Conflicts

Two revisions of a file that contain conflicting content can be merged. Use the procedure to merge two revisions with conflicting content.

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 Applications window.)
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).

Using the Subversion Merge Wizard

Merge tracking in Subversion records the merges that you completed. In the Merge Wizard, you can select specific revisions, branches, or change sets to merge.

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.

Working with File Versions and History in CVS

You can merge, compare, replace, and view different versions of files in the CVS repository.

CVS gives you techniques that allow you to merge, compare, replace, and view different versions of the files in your repository.

Merging Files in CVS

In CVS, two revisions of a file that contain conflicting content can be merged. Use the procedure to merge two revisions with conflicting content.

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.

Comparing Files in CVS

Revisions of files that are under CVS source control can be compared with the preceding revision of the file, or with earlier revisions. Use the procedure to compare a file shown in the Applications window.

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 Applications window:

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.

Replacing a File with a CVS Revision

In CVS, replace a file with the base or head revision, or with a file with a specific numbered or tagged revision. Use the procedure 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 Applications window, 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**.

Viewing the History and Status of a File in CVS

Knowing the history and status of a file helps a user modify or update the file. Use the procedure to view the status of a file, and use the step to view the history of a project or 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 Applications window, choose **Team > 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 Applications window, open the context menu and select **Team > 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.

Working with File Versions in Perforce

Use Perforce tools to resolve conflicts in file versions because conflicts prevent the submission of files. Use the procedure to revert to a non-conflicting file version.

Perforce provides tools for resolving 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.

To revert to a non-conflicting file version:

- Select the file in the Applications window and choose **Team > Revert**.

To resolve conflicting file versions (assumes use of Perforce merge tool):

1. Open the Perforce client by choosing **Team > 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.

Working with File Versions in Team System

Use Team System tools to view, compare, and resolve conflicts in files in the repository.

Team System provides tools for working with file versions, including viewing, comparing, and resolving conflicts among different files in the repository.

Resolving Conflicts in Team System File Versions

If Team System displays a system message about file conflict, either resolve the conflict, or revert to a non-conflicting version of the file. Use the step to revert to a non-conflicting file version.

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 Applications window and choose **Team > Undo**.

Undoing Changes to Team System Files

Undo the changes made to a file using the Undo option in the Team menu.

Use to undo the most recent change to a file.

To undo changes:

1. In the Applications window, select the file whose last change you want to undo.

2. Select **Team > Undo**.

   The Undo dialog is opened.

   The change will be undone when you click **OK**.

Replacing a File with the Team System Base Version

In Team System, you can replace a file with the base version, the source of the working copy of the file. Use the procedure to replace a file with the Team System base revision.

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 Applications window, 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**.

Viewing the History of a File

History Viewer displays the history of files under Team System control. Use the step 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 Applications window, choose **Team > Version History** from the context menu.

For more information while using the History Viewer, press **F1**.

Comparing Files In Team System

In Team System, compare files with other versions of the file or with other files. Use the procedures to compare versions of a file, to compare a file with another file, and to compare two files.

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 Applications window.
2. From the context menu for one of the files, choose **Compare With > Each Other**.

The files are shown in the Compare tool.

Using an External Diff Tool with CVS

JDeveloper supports External Tools to integrate external compare viewers. Use the procedure to integrate CVS DIFF.

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**.

Integrating a Third Party Diff Utility

Use utilities such as Araxis Merge to view the differences between two revisions of a file in the history tool. Use the procedure 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.

To integrate a third party diff utility:

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**.

Integrating other CVS Commands

Other CVS commands can be integrated into JDeveloper using external tool macros. Use the procedure to integrate other 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.

To integrate other commands

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**.

Working with Patches in Source Control

Create and apply patches in JDeveloper using version control systems. Use the procedures to create and apply a patch.

Many of the version control systems available in JDeveloper provide 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.

Subversion (included in JDeveloper) uses the following procedures for creating and applying patches. In addition, the version control extensions for Concurrent Version System (CVS), Perforce and Team System all use the same steps for creating and applying patches.

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 Applications window, 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 **Team > 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**.

How to Create and Apply Patches

Create and apply patches in JDeveloper using version control systems. Patches can be saved as separate files, or copied-and-pasted as sections into target files.

JDeveloper 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. You can typically save these patches either as a separate file, or as a section of text to be copy-and-pasted into the target file.

Creating Patches

Use a patch to record the changes between two revisions of a file, and to apply the differences to a third file. Use the procedure to create a patch.

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**.

Applying Patches

Apply a patch that you created using the Apply Patch option from the Team menu. Use the procedure to apply a patch.

Once you have created a patch, you apply it from the Team menu.

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 **Team > 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**.

7

Getting Started with Developing Java Applications

This chapter is an overview of the tools and features that JDeveloper provides to speed up the process of writing Java code.
This chapter includes the following sections:

- About Developing Java Applications
- Using the Java Source Editor
- Using Code Insight
- Using Code Peek
- Using Scroll Tips
- Using InfoTips
- Searching Incrementally
- Using Shortcut Keys
- Bookmarking
- Browsing Java Source
- Using Code Templates
- Setting Preferences for the Java Source Editor
- Using Toolbar Options
- Using the Quick Outline Window
- Working with the Java UI Visual Editor

About Developing Java Applications

From the JDeveloper IDE, use Swing and AWT components to build, deploy, and publish desktop-based GUI applications, and use Java Web Start to build Java client 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.

Using the Java Source Editor

JDeveloper displays Java source files in the Java Source Editor. Use the editor to edit Java code, and use the Preferences Dialog to customize the 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.

Double-clicking a node in the Applications window 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.

Using Code Insight

Use Java Code Insight to filter irrelevant information, to speed up coding, to complete code using completion insight, to select instance and static members using member insight, and to edit parameters using parameter insight. Use Preferences from Tools to edit Code Insight settings, or to view or change accelerators.

With Java Code Insight, you can filter out unnecessary information such as top-level packages, imported classes, default Object methods, deprecated items, and emphasize more key detail such as local variables, locally declared members, overloaded methods.

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 keymap ping, 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. The 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.

You can also use and 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.

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.

Using Code Insight to Add Annotations to Your Java Code

An annotation associates information with a program element. When you add an annotation, use Member Insight for a list of options based on the statement context, and use Parameter

Insight for annotation information such as the names of the elements of the annotation type, the default values, and the created values.

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.

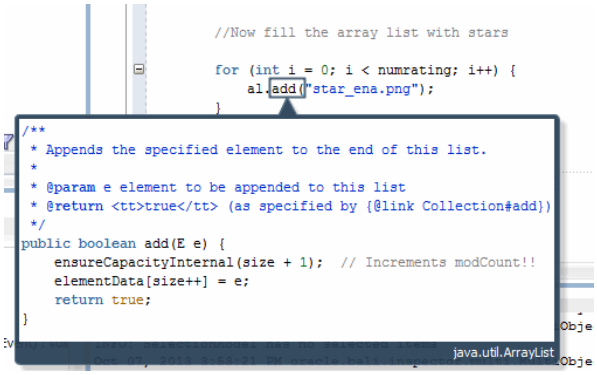
When you start adding an annotation, Member Insight (Ctrl+Space) displays a list of options (fields, members, classes) based on the statement context. Parameter Insight (Ctrl+Shift+Space) displays information about the annotation like the names of the elements of the annotation type, the default values, and the created values. It also highlights the element currently under the cursor in the annotation.

Using Code Peek

Press Shift and point the mouse pointer at a variable or method to see its definition.

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.

Figure 7-1 Code Peek



Using Scroll Tips

Scroll vertically to see a window alongside the scroll bar that displays the methods that are in view and the name of the method that will next appear in view.

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.

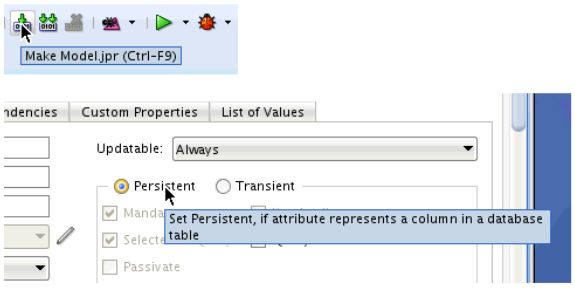
Using InfoTips

An InfoTip, with details of the item in focus, pops up when the mouse pointer is positioned over an item.

An InfoTip is a pop-up window that reveals additional detail about an item in the application, for example, text in the Java Source Code editor and the overview gutter. You display the InfoTip

by pointing or hovering over the item with your mouse cursor. It remains visible until you move the mouse away or click elsewhere in the UI.

Figure 7-2 InfoTips



Searching Incrementally

Use the Incremental Find Forward or the Incremental Find Backward options in Search to search for an item forward or backwards.

To search incrementally, from the main menu choose **Search** > **Incremental Find Forward** or **Search** > **Incremental Find Backward**. Begin typing in the dialog that displays. 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.

Using Shortcut Keys

Shortcut keys or accelerators are a keyboard alternative to mouse device operations and navigation. Use Preferences in Tools to go to the Shortcut Keys page to view or change existing accelerators, and to the Code Editor from the Category list to view or change accelerators for an editor.

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.

Bookmarking

Use Bookmark to see all the bookmarks in the Bookmarks window. The window also shows the line number and the name of the bookmarked method.

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 by pressing Ctrl+<*number*>.

Browsing Java Source

Browse imported classes and interfaces, member fields and methods, and local variables using Go to Declaration that takes you to the identifier source in an open Java file. JDeveloper reverse-engineers the class file for a source that is unavailable.

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 following code revokes the declaration search at SetText. It brings up the source code for `javax.swing.JButton`, with the `SetText()` method displayed.

```
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.

Using Code Templates

In application development, use Code templates to insert pre-written code that adapts to surrounding code. Use Preferences in Tools to create and edit a code template, and view the complete list of code templates in the Code Editor Help.

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. See How to Use Code Templates.

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.

Setting Preferences for the Java Source Editor

Use Preferences in Tools to customize the behavior of the Java Source Editor, and to define the settings for the general source editing environment. Use the procedure to set the options for Code Insight as it applies to Java.

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. See How to Set Preferences for the Source Editor.

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.

   A list of selectable options displays, for example, Bookmarks and Code Templates.
3. Click an option and choose options that appear on the new page that displays.
4. Click **OK** when you are finished.

How to Set Comment and Brace-Matching Options for the Java Source Editor

Set comment and brace-matching options for the Java source editor. Use the procedure to set the options for Java comment and brace matching in the source editor.

JDeveloper enables you to set comment and brace-matching options for the Java source editor. For example, you can choose to add leading and closing asterisks as you add new lines either in multi-line Java comments or in Javadoc comments.

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.

   For example, you can select the **Add Closing Bracket or Parenthesis Automatically** option.
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.

How to Set Import Statement Sorting Options for the Java Source Editor

Set the options to sort import statements in the Java Source Editor. Use the procedure to set the options for sorting import statements.

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, choose options to create the behavior that you want when sorting import statements in the editor.

5. Click **OK**.

How to Choose a Coding Style

Use Preferences in Tools to select a profile that decides the coding style which determines conventions for the Java Code shown in the Java Source Code editor. Use the procedure to select a coding style for the Java Source Code editor.

You can use the **Tools** > **Preferences** dialog to select a profile that determines the style of code shown in the Java Source Code editor. The coding style determines Java Code conventions such as formatting, how import statements display in code, variable names, and member order.

To select a coding style for the Java Source Code editor:

1. From the main menu, choose **Tools** > **Preferences**.
2. In the Preferences dialog, expand the **Code Editor** node.
3. Click the **Code Style** node.
4. On the Code Style page, choose a **Profile**.
5. Click **OK**.

How to Edit a Coding Style

Edit a coding style and make it available as a selectable Profile option on the Code Style page. Use the procedure to edit a code profile.

You can edit an existing coding style and make it available as a selectable **Profile** option on the Code Style page. You can also import an existing code style profile from an XML file.

To edit an existing code profile:

1. Choose an existing profile in the **Profile** list.
2. Click **Save As** and enter a name for your new profile.
3. Click **Edit**.
4. In the Edit Code Style Profile - Java Code Conventions page, choose options for your new profile.
5. Click **OK**.
6. You can alternately click **Import** to import a code style profile into JDeveloper from an XML file.

How to Set Up a Coding Style Using an Extension

Use an extension to create a custom coding profile that determines coding style. Use the procedure to set a default coding profile using an extension.

You can create a custom coding profile using an extension, then set it as the default for a role, for example, developer, using the `extension.xml` file. See Introduction to Developing Oracle JDeveloper Extensions and How to Edit a Coding Style.

To set a default coding profile using an extension:

1. Edit a custom style.
2. Choose **Export** from the **More Actions** list.

3. Add it to an extension and put an entry in the `extension.xml` file, similar to the following.:

```
<hooks>
  <code-style xmlns="http://xmlns.oracle.com/ide/extension">
    <profile-url>/META-INF/custom-profile.xml<profile-url>
  </code-style>
<hooks>
```

4. To set the profile as the default style, add a section to the role shaping file with the profile name, similar to the following:

```
<c:settings-ui-customizations>
  <c:page idref="/preferences/ceditor/style">
    <c:field idref="CodingStylePreferences.profile">
       <c:value>Custom Profile</c:value>
    </c:field>
  </c:page>
</c:settings-ui-customizations>
```

Using Toolbar Options

The Java Source Editor displays Java source files and the tools to create and edit Java code. The table lists the Toolbar options that JDeveloper provides.

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 7-1.

Table 7-1 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 (See 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.
	Surround	Click to surround the currently selected block of text in the Java Source Editor with a coding construct, using the Surround With dialog.
	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.

Table 7-1 (Cont.) Toolbar Options

Icon	Name	Description
	Reformat	Click to apply source formatting to your code.
	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.

Using the Quick Outline Window

In the Java source editor, click Quick Outline Toolbar to see the Quick Outline window. The window provides a structural view of the available methods and fields of a class and its super classes.

Clicking the **Quick Outline Toolbar** icon to the right of the **Find** field in the Java source editor displays the Quick Outline window shown in Figure 7-4.

Figure 7-3 Quick Outline Icon

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 place.

Figure 7-4 Quick Outline Window

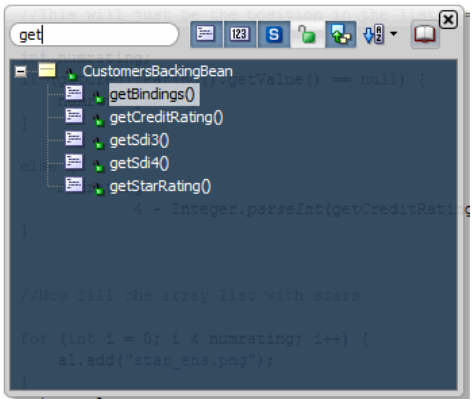


Table 7-2 lists the available icons and options.

Table 7-2 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
	Show Detail Window	Click to open the Show Detail window for the selected item in the Quick Outline window. In the Show Detail window, click the Documentation tab to view the package containing the item. In the Show Detail window, click the Code tab to see its definition in the source file.

Working with the Java UI Visual Editor

For Swing or Abstract Window Toolkit applications, use the Java UI Visual Editor to view and edit the visual components of the user interface.

The Java UI Visual Editor displays the visual components of a user interface in editing mode. See Editing with the Java Visual Editor .

> **Note:**
>
> You can use the Java Visual Editor for Swing/AWT Applications only.

Java Swing and AWT Components

Java Swing and AWT components are the basic building blocks, such as buttons, dialogs, and menus of an application UI. To build the UI, in the Java Visual Editor, select JavaBeans from the Components window, set the values of the component properties, and attach `eventhandler` code to the component events.

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 Components window, 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.

See About Java Swing UI Components and Containers.

8

Working with Java Code

This chapter describes how to take advantage of the JDeveloper tools and features that help you create the code for your Java applications. For example, you can browse Java elements in your application using a JDeveloper dialog or directly from an open file in the Java Source Editor. To edit Java code directly, you can use the Source Editor. Use the Visual Editor to display a diagram representing your Java code. For example, a Frame component displays as rectangular box, a button displays as a smaller button on top of the Frame, and so on. **Refactoring** is an editing technique that modifies code structure without altering program behavior. Refactoring is useful when you want to modify a program's source code to make it easier to maintain, extend, or reuse.

This chapter includes the following sections:

- About Working with Java Code
- Navigating in Java Code
- Editing Java Code
- How to Add Documentation Comments
- Refactoring Java Projects

About Working with Java Code

In the Java Source Editor, write or edit Java code with predefined code templates, add Javadoc comments, and apply formatting to code. Access the features from the context menu and the Source main menu.

In most cases, you use the Java Source Editor to write or edit Java code. It contains a set of Java-specific features to enhance your coding experience. For example, JDeveloper makes it easy to code using predefined code templates, add Javadoc comments, and even apply formatting to your code. These features are available through the context menu and the Source main menu. See Using the Source Editor.

You can customize the behavior of the Java Source Editor by specifying preferences in the Preferences Dialog. See How to Set Preferences for the Source Editor.

Navigating in Java Code

JDeveloper provides Java-aware editing features such as moving to the source code of classes and interfaces and their members, and navigating with keyboard accelerators from member to member in a class definition in the Java Source Editor.

JDeveloper provides many Java-aware editing features that you can use to improve your productivity. These include features for locating and moving to the source code for your projects' classes and interfaces and their members. JDeveloper also comes with keyboard accelerators to step from member to member in a class definition in the Java Source Editor.

How to Browse Java Elements

Browse Java elements either from a JDeveloper dialog or from a file that is open in the Java Source Editor.

While working in JDeveloper, you can browse Java elements using a JDeveloper dialog or directly from an open file in the Java Source Editor. You also can locate source code or Javadoc for a given type.

Sources are displayed for classes present in the project's source path. They are also displayed for classes present in a project library, if the library has sources included.

Browsing a Java Element Directly in JDeveloper

Use the Go to Java Type Dialog to browse Java elements. Use the procedure to browse a Java element in JDeveloper.

You can browse Java elements in the JDeveloper dialog.

To browse a Java element in JDeveloper:

1. From the main menu, choose **Navigate** > **Go to Java Type**.

   You can also use the keyboard shortcut, Ctrl+minus.

2. In the Go to Java Type dialog, enter the name of the Java class or interface that you want to locate.

   When you begin entering text in this field, a list of Java entities matching the text displays. For example, entering **j** might display a list of classes that includes `java.lang`.

   You can drill down in the results returned in the Go to Java Type dialog. For example, double click on the **java.lang** class to display methods such as `java.lang.Byte`.

3. Double-click an entity in the list to open it in the source editor.

How to Locate the Declaration of a Variable, Class, or Method

In the Java Source Editor, locate the declaration of any identifier. Follow the steps to navigate to the declaration of a code element.

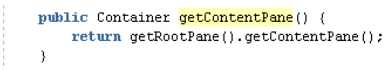
When working in the Java Source Editor, you can quickly locate the declaration of any identifier.

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.

Figure 8-1 Highlighted Declaration



If the declaration is in the same file, the cursor moves to it and highlights it. If the declaration is in a different file, the source file opens in the Java Source Editor. The cursor moves to it and highlights it.

How to Find the Usages of a Class or Interface

In the Java Source Editor, search for references to a class or interface and its members. Use the procedure to find the usages of a class.

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 are reported. You can extend the search to libraries if the source files for the libraries are accessible.

To find the usages of a class:

1. Select the class or interface in one of the following ways:
   - In the Java Source Editor, select the name.
   - In the Applications window or the Structure window, select a class or an 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 displays.
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. To 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, are discarded.
6. Click **OK**.

   The results display in the Usages Log window.

How to Find the Usages of a Method

In the Java Source Editor, search for references to a method. Use the procedure to find usages of a method to instances of the class or interface for which the method is defined, or usages of its subclasses or subinterfaces that inherit the method.

While working in the Java Source Editor, you can quickly locate references to a method.

The search displays applications of the method to instances of the class or interface for which the method is defined. It also shows applications for instances of its subclasses or subinterfaces, if any, that inherit the method.

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 method.
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.

How to Find the Usages of a Field

In the Java Source Editor, search for references to a field in instances of the class or interface for which the field is defined, and instances of its subclasses or subinterfaces that inherit the field. Use the procedure to find the usages of a field.

While working in the Java Source Editor, you can quickly locate references to a field.

The search shows references to the field in instances of the class or interface for which the field is defined. It also shows references for instances of its subclasses or subinterfaces, if any, that inherit the field.

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.

How to Find the Usages of a Local Variable or Parameter

In the Java Source Editor, search for references to a local variable or a parameter in a method body. Use the procedure 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. Local variables and parameters used in extracted code become parameters of the new method.

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 results are displayed in the Usages Log window.

Identifying Overridden or Implemented Method Definitions

In the Java Source Editor, methods that override or implement superclass definitions can be identified. Click Overrides or Implements in the margin to view the overridden definition of a method.

While working in the Java Source Editor, you can identify methods that override or implement superclass definitions. Overriding definitions are marked with the **Overrides** up arrow icon .

Overridden definitions are marked with the **Implements** margin icon , in the Java Source Editor margin.

To view the overridden definition of a method, click the **Overrides** margin icon. To view the overridden definition of a method, click the **Implements** margin icon.

Click the **Back** button on the Main toolbar to return to the previous view.

How to View the Hierarchy of a Class or Interface

In the Java Source Editor, inspect the hierarchy of subtypes and supertypes 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.

What displays in the editor is the entire GUI hierarchy for the node. The method of display depends upon whether this hierarchy consists of menu or non-menu items.

Viewing the Hierarchy of a Class or Interface in the Java Source Editor

View the hierarchy of selected classes or interface in the hierarchy window. Use the procedure to view the hierarchy of a class or interface in the Java Source Editor.

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 **Types** or choose **Navigate** > **Types**.

   The Types window opens (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.

Stepping Through the Members of a Class

In a Java Source Editor, use keyboard accelerators to move from member to member in a class definition. Use the code-stepping commands listed in the Navigate category of the

Shortcut Keys page of the Preferences dialog or the Navigate options to navigate between members.

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**.

The following code-stepping commands are also defined but are not assigned default accelerators and are not available through the **Navigate** menu:

- **Go to Next Class**
- **Go to Next Field**
- **Go to Next Method**
- **Go to Previous Class**
- **Go to Previous Field**
- **Go to Previous Method**

You can find these commands listed in the **Navigate** category of the Shortcut Keys page of the Preferences dialog. You can add or change accelerators.

Editing Java Code

JDeveloper provides Java-aware editing tools such as the Java Visual Editor, Source Editor, and the Visual Editor.

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.

How to Create a New Java Class or Interface

Create an application, a project, a new class, or an interface that JDeveloper adds to an active project, and displays in the Java Source Editor. Use the procedure to create a new class, and add it to a project.

Before creating a new class or interface, you must first create an application and a project. After you create the class or interface, it is added to the active project and, by default, displays in the Java Source Editor. You can also access it in the Applications window.

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

1. In the Applications window, select the project where you want to add the Java class, for example, **client**.
2. Right-click and choose **New** > **Java Class**.
3. In the Create Java Class dialog, enter the class or interface name, the package name, and the superclass that the new class will extend.
4. Select attributes as needed.

5. Click **OK**.

How to Implement a Java Interface

In the source editor, add framework code that modifies a target class to implement an interface, or make a target interface extend another interface. Use the procedure to implement an 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**.

How to Override Methods

In the source editor, add stub definitions to a class to override methods inherited from superclasses. Use the procedure to override methods.

In the source editor, you can quickly add stub definitions to a class to override methods inherited from superclasses. An overriding subclass provides a specific implementation of a method that is already provided by one of its 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. These must be implemented by non-abstract types.
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.

How to Convert an Anonymous Inner Class to a Lambda Expression

Lambda expressions provide a mechanism to express instances of anonymous inner classes that contain only a single method. Use the procedures to convert an inner class to a lambda expression, and to turn off the feature.

Lambda expressions enable you to simplify coding by providing a mechanism for expressing instances of anonymous inner classes that contain only a single method in more compact forms. You can only convert such a class only if it meets the requirement of being a functional interface, that is one whose type can be used for a method parameter when a lambda expression is used as the argument to the method.

> **Note:**
>
> This feature is only accessible if you have specified JDK 1.8 as the target JDK for your project.

To convert an inner class to a lambda expression:

1. In the Java source editor, select the anonymous class to convert.

   The IDE analyzes the code to check if code assist is applicable.
2. Click the code assist icon that appears in the editor margin.
3. Click the code assist **Convert Anonymous Inner Class to a Lambda Expression**.

   The lambda expression replaces the inner class.

See `http://docs.oracle.com/javase/tutorial/java/javaOO/lambdaexpressions.html`

You can turn this feature on or off in the Preferences dialog. Note that this feature is on by default.

To turn this feature off:

1. Go to **Tools** > **Preferences** and click the **Audit** node.
2. Click **Edit Profiles** on the **Audit** page.
3. Click the Code Assists tab.
4. Expand the following Code Assists category nodes: **Java SE** > **Java** > **Code Assists**.
5. Uncheck the **Eligible Lambda Expression** to make the feature unavailable.

   Select to make the feature available through code assist.

How to Use Code Templates

Use the predefined code templates in JDeveloper to insert text for commonly used statements. Use the procedure to evoke a defined code template.

JDeveloper provides predefined code templates that 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 (Object object : Object) {
            ;
        }
```

> **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.

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.

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. See How to Customize Code Templates for the Source Editor.

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. Enter 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.

Using Predefined Code Templates

Use the predefined JDeveloper code templates in the list to insert code for commonly used statements.

The predefined code templates that JDeveloper provides are shown below.

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

```
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 ($end$)
{

}
```

How to Expand or Narrow Selected Text

Use the Expand or Narrow Selection option to expand or narrow a selected block of code. Use the procedure to expand selected code.

You can use the **Expand/Narrow Selection** option to successively expand or narrow a selected block of code, based on Java syntax. 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.

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.

Use the **Narrow Selection** option (or press Ctrl+Shift+Minus) to successively reduce selected code in the same fashion.

How to Surround Code with Coding Constructs

In the Java Source Editor, use coding constructs to surround Java statements and blocks. Use the procedure to surround a block of code with a construct.

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**. This options is available by right-clicking in the Visual Editor and choosing **Surround** from the context menu. Alternatively, you can click the **Surround** ({}) icon on 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.

Code constructs can be set using the Code Templates page of the Preferences dialog. You can customize code templates as needed or modify existing ones. See How to Customize Code Templates for the Source Editor.

How to Fold Code

In a file open in the Java Source Editor, code folding collapses selected blocks of code so that other areas of code can be viewed better. Follow the steps to use code folding.

You can use code folding to hide and display sections of a file currently open in the Java Source Editor. Code folding can improve readability, letting you view specific areas by folding selected blocks of code, such as function definitions.

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 +.
- Click on the + sign to unfold the code, displaying the full contents of the area you previously folded.

Right-click between the signs in the margin to open a context menu from which you can select commands to expand or collapse specific areas of code throughout the entire file.

If you have code inside a method such as that shown below, you can collapse the middle chunk of code so you do not have to see it when working on another part of the method.

```
public void main(String[] args) throws SQLException, IOException {

    //... some code ...

    {
      Run.dbgPrnt("Extractor main() querying => BSN");
      // make the basin file
      query = "select * from BSN";
      rset = OracleAccess.makeResultSet(query, stmt);
      rset.next();
      l = Basin.extract(rset, Version);
      Format.writeFile(outPath, "groupings.txt", l);
    }

//    ... some more code ...

  Run.dbgPrnt("Extractor main() has ended");
}
```

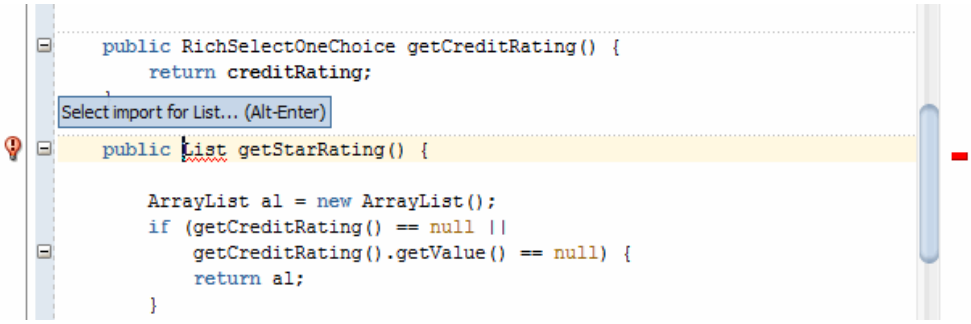
You can fold code for inner code blocks such as for, while, switch, {...}, etc.

Adding an Import Statement

Any unresolved reference to an unavailable class in the Source Editor causes JDeveloper to import the class, or display possible matches for selection so that an import statement can be added. Set Import Statement Options in the Java Source Editor to configure or disable Import assistance.

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.

Figure 8-2 Import Statement Needed



JDeveloper automatically adds 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 displays all possible matches from the class path. You 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 does not recognize a class, a light bulb 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.

How to Organize Import Statements

Organize import statements in the Java Source Editor by setting the options for organizing imports in the Preferences dialog. Follow the step to organize import statements in a source file.

You can organize import statements easily in the Java Source Editor. Set the options for organizing imports 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.

To organize import statements in a source file:

- With the file open in the editor, right-click and choose **Organize Imports**.

Using ojformat

Use the `ojformat` command line tool that is located in <JDeveloper_Home>/jdev/bin to reformat workspaces or projects.

ojformat is a command line tool that you can use to reformat workspaces or projects. It is located in `<JDeveloper_Home>/jdev/bin`.

The syntax for **ojformat** is

```
ojformat option file
```

where

option is an option such as `-ade`. `-ade` indicates that ADE extension for version control should be loaded. Other version controls should work automatically.

Examples

```
ojformat -ade application.jws
```

Reformats all projects in application.jws. Version control is ADE.

```
ojformat application1.jws application2.jws
```

Reformats all the projects in both applications

```
ojformat application.jws project1.jpr
```

Reformats `project1.jpr` of application.jws

```
ojformat app1.jws project1.jpr app2.jws project2.jpr
```

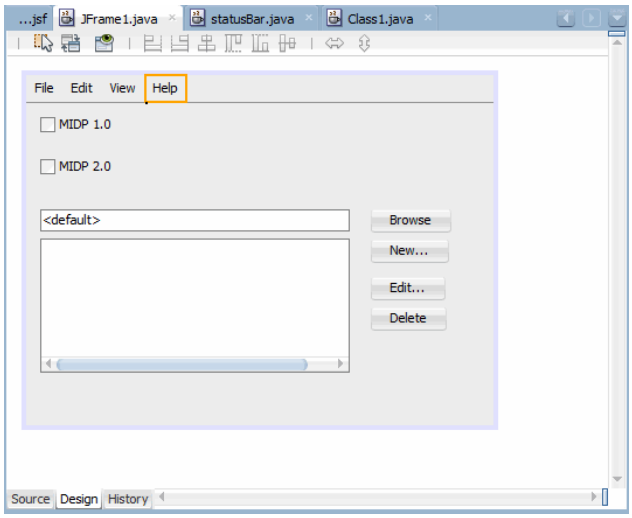
Reformats project1 of app1 and project2 of app2

Editing with the Java Visual Editor

The Java Visual Editor displays the visual components of a user interface in Design so that you can view and edit source code as you design the UI. Use Source in Source Editor to access the source code.

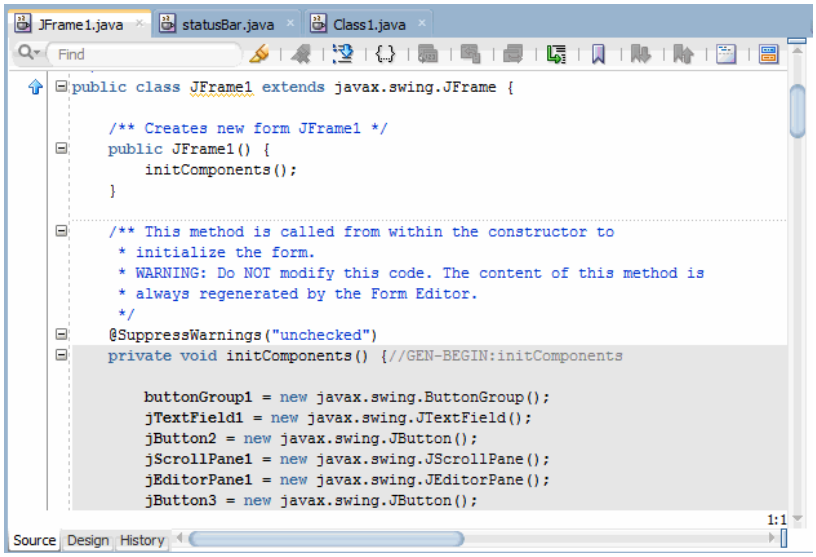
The Java Visual Editor displays the visual components of a user interface in the Design tab, as shown in Figure 8-3.

Figure 8-3 Java Visual Editor Design View



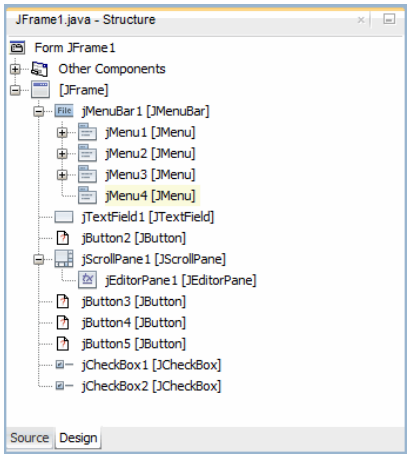
You see the Java source view of the visual classes by clicking the Source tab as shown in Figure 8-4.

Figure 8-4 Java Visual Editor Source View



When the Java Visual Editor is open, its corresponding elements are displayed hierarchically in the Structure window, as shown in Figure 8-5.

Figure 8-5 Structure Window for Java File



If the Properties window is open, selecting elements in either the Structure window or the Java Visual Editor changes the selection in the Properties window as well.

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 non-menu or menu items, and the commands available within the context menu depend on the selected object.

To open the Java Visual Editor:

- Double-click the Java file in the Applications window and click the Design tab in the editor window.

The source code is accessible in the Source Editor (click the Source tab to view the source code), enabling you to view and edit your source code in parallel with designing your UI. Any changes made in the Java Visual Editor or Properties window are immediately reflected in the source code.

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.

For information on working with the Java Visual Editor, see Implementing Java Swing User Interfaces .

Protecting Code

JDeveloper supports write protection for code that changes by preserving code that is required for its functioning.

JDeveloper provides write protection for code that you are changing. It does this to preserve the code it requires to function. Figure 8-6 shows a section of Swing/AWT code that is colored grey, indicating that this section of code is protected.

Figure 8-6 Protected Code

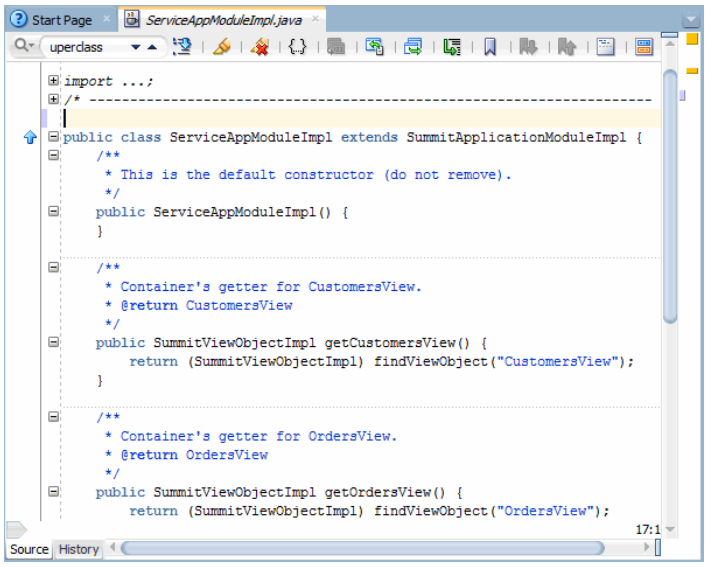
```
 */
public class NewJPanel extends javax.swing.JPanel {

    /** Creates new form NewJPanel */
    public NewJPanel() {
        initComponents();
    }

    /** This method is called from within the constructor to
     * initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is
     * always regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
  private void initComponents()
  {

    javax.swing.GroupLayout layout = new javax.swing.GroupLayout(this);
    this.setLayout(layout);
    layout.setHorizontalGroup(
      layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
      .addGap(0, 400, Short.MAX_VALUE)
    );
    layout.setVerticalGroup(
      layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
      .addGap(0, 300, Short.MAX_VALUE)
    );
  }


  // Variables declaration - do not modify
  // End of variables declaration

}
```

How to Add Documentation Comments

Create and maintain Javadoc comments using the editing commands in JDeveloper. Follow the steps to add documentation comments to a source file.

You can use JDeveloper's editing commands to create and maintain Javadoc comments, such as those in Figure 8-7. After you enter Javadoc comments, you can use the Find in Files dialog to search for them.

Figure 8-7 Javadoc Comments



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. Or,
- With the code element selected in the Structure window, choose **Source** from the main menu, then **Add Javadoc Comments**.

A template for the documentation comment is inserted into the file. Add information to the template to describe the element.

How to Update Documentation Comments

In the Java Source Editor, record changes that are made to elements by adding or removing tags from documentation comments, and adding tag descriptions. Use the procedure to update documentation comments in a source file.

You can update documentation comments in the Java Source Editor. Tags are added to or removed from the documentation comment to reflect changes you have made to the element. Add descriptions for the new tags.

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**.

How to Set Javadoc Properties for a Project

JDeveloper provides project defaults that you can modify for individual projects or for all projects across workspaces. Use the procedure to set Javadoc properties for an individual 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 Applications window, select the project.
2. From the main menu, choose **Application** > **Project Properties**, or right-click and choose **Project Properties**.
3. Choose **Javadoc**.
4. Set attributes.
5. When finished, click **OK** to close the Project Properties dialog.

How to Customize Documentation Comment Tags

Customize usage of documentation comment tags in the Java code editor by creating custom tags, associating the tag with code elements, defining it for inclusion in a project, assigning it a default value, and giving it an order in the tag list. Use the procedure to define a custom tag.

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.

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 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 the Plus (+) icon.

   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**.

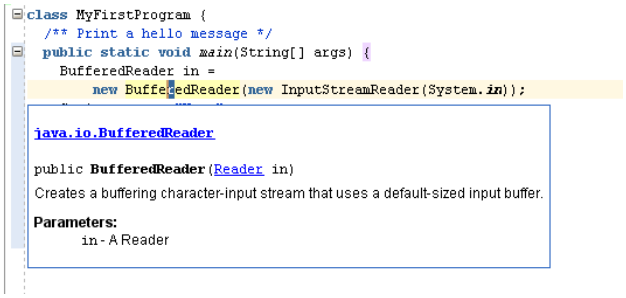
How to View Javadoc for a Code Element Using Quick Javadoc

In the Java Source Editor, when the selected source code satisfies certain criteria, the Quick Javadoc feature becomes available so that you can use Javadoc comments for classes, interfaces, or individual members. Use the procedure to display Javadoc for a code element.

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.

As shown in Figure 8-8, the Quick Javadoc feature looks up the selected entity on the source path and displays the Javadoc comment entered in a popup window.

Figure 8-8 Quick Javadoc Window



If no Javadoc comment exists for that element, an empty Javadoc comment is displayed.

The Quick Javadoc feature is available when the selected source code meets the following criteria. It is:

- On this project's source path.
- On the source path of a project that the current project depends on.
- Available for a library assigned to this project
- A part of the JDK in use.

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.

How to Preview Documentation Comments

The Quick Javadoc feature allows you to preview documentation comments in source files. Use the procedure to display documentation comments for a given class, member, or method call.

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.

How to Audit Documentation Comments

Use the Audit Profile and Audit filename options for Javadoc Comments to validate documentation comments in source files. Use the procedure to check documentation comments in a source file.

You can validate documentation comments in your source files. The audit reports formatting errors and missing or extraneous tags.

To check documentation comments in a source file:

1. In the Applications window, 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 to configure the audit to choose which types of errors to search for or to ignore, click **Edit**.

   The Audit Profile dialog opens with the Rules tab selected.
5. Expand **JavaSE** > **Java** > **Javadoc Comments** nodes and select one or more Javadoc rules. Select an item to see a description of the validation check. For each item, you can set the property values for each rule.
6. Click **OK** in the Audit Profile dialog.
7. Click **Run** in the **Audit *filename*** dialog.

   The results of the audit appear in the Log window.

How to Build Javadoc

Set properties for a project on the Javadoc page of the Preferences dialog to generate API references and other documentation from the Applications window. Use the procedure to build Javadoc on a package, file, or project.

You can generate API references and other documentation directly from the Applications window, 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 build Javadoc on a package, file, or project:

1. Select the appropriate node in the Applications window.
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.

How to Create References to Missing Annotation Elements

When auditing unresolved Java annotations in the Java Source Code editor, create references to missing annotation elements. Use the procedure to create a reference to a missing annotation.

You can create a reference to a missing annotation element when displaying an audit hints for an unresolved Java annotation in the Java Source Code editor.

To create a reference to a missing annotation:

1. Open the Java class in the Java Source Code editor.
2. Enter the annotation in the Java class source code, for example, `@stateless`.
3. Hover your cursor over the annotation.

   A dialog displays indicating that the type, for example, `@stateless`, cannot be found.
4. Click the **More** link.
5. Click the **Create Annotation Type** <***type***> link.
6. In the Create Annotation Type dialog, enter the name of the package where you want to locate the annotation class, for example, **project1**.
7. Click **OK**.

Using the JOT Structure Window

The Java Object Tree structure window displays the hierarchy of a source element that is selected in the Java Source Editor. Install the Java Object Tree Structure extension using Check for Updates in Help, and get the extension window displayed by selecting JOT Structure Pane from the Window options.

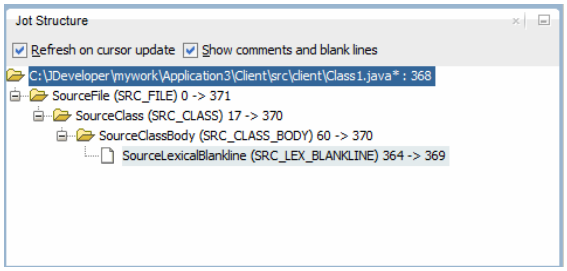
The Java Object Tree (JOT) structure window displays a hierarchical tree of a source element selected in the Java Source Editor. For example, your Java file might contain code similar to the following:

```
public class Class1 implements Serializable{
    @SuppressWarnings("serial:-4347721670334465144")
        // test
    private static final long serialVersionUID = -4347721670334465104;
    public Class2() {
        super();
    }
    /* Non-javadoc comment */

    public void a(){}
    // @SuppressWarnings("serial:23")
```

```
    private class foo{}
}
```

Based on this, the JOT structure window displays at the bottom of JDeveloper, as shown in Figure 8-9. The hierarchy contains all of the child and parent source elements in the code. This could be the class, code blocks, or other elements.

Figure 8-9 JOT Structure Window



If you select a SourceElement in the Java Source editor, the corresponding SourceElement, including corresponding text, is highlighted in the JOT Structure window.

The Java Object Tree Structure is a JDeveloper extension that you must install to access this feature. Go to **Help** > **Check for Updates** to install the extension.

Once installed, display the JOT Structure window by choosing **Window** > **JOT Structure Pane**.

How to Display Comments and Blank Lines in the JOT Structure Window

The JOT Structure Window displays Java and non-Java comments and blank lines in Java code. Use the procedure to display comments and blank lines.

In the Java structure window, you can display Java and non-Java comments and blank lines in your Java code. For example, if **Show comments and blank lines** is selected and you set your cursor on a blank line in your Java code, the blank line displays in the tree with corresponding text, for example, `SourceLexicalBlankline (SRC_LEX_BLANKLINE).`

To display comments and blank lines:

1. In the Main menu, choose **View** > **JOT Structure Window**.
2. Select **Show comments and blank lines**.

How to Set the Refresh Mode in the JOT Structure Window

You can use the **Refresh on cursor update** checkbox to allow refresh of the hierarchical tree on or off.

For example, with the checkbox selected, you might place your cursor on some text in your Java source file. The SourceElement of that text and its parents in the JOT tree display. In order to freeze the state of the JOT Structure Window and develop some code that uses this particular SourceElement, turn off the **Refresh on cursor update** checkbox. Load the Java source file you want to change. While you work, you can refer to the frozen state of the JOT tree.

To set the refresh mode:

1. In the Main menu, choose **Window** > **JOT Structure Window**.
2. Select **Refresh on cursor updates**.

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 re-validate the program to ensure that no errors have been introduced.

Table 8-1 contains some examples of simple refactoring operations.

Table 8-1 Refactoring Operations

Operation	Description
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.
Changing a schema's target namespace	All the referring schemas are updated to have the new target namespace.

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.

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 re-factored 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 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 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)

How to Invoke a Refactoring Operation

JDeveloper provides a wide range of automated refactoring operations that enable you to enhance code quality. The external behavior of the code is not altered, yet its internal structure improves.

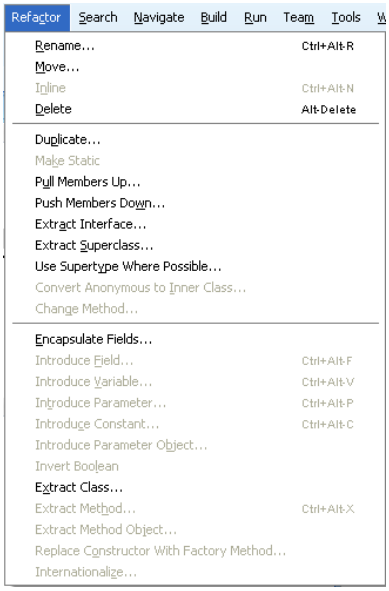
To invoke a refactoring operation:

1. Select a program element in a source editor window, Applications window, or structure pane.
2. Right-click on the program element and choose **Refactor**.
3. Choose an operation from the context menu, for example, **Rename**, **Move**, or **Delete**.

You can also choose **Refactor** from the toolbar and select a refactoring operation from the drop-down list.

As shown in Figure 8-10, refactoring context menus contain different items depending on where in JDeveloper you are right-clicking to display the menu.

Figure 8-10 Refactoring Drop-down List



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.

How to Preview a Refactoring Operation

A preview provides a view of usages so that they can be inspected and resolved before they are committed. Use the procedure to commit a refactoring operation.

When performing a refactoring operation that may modify many usages, it is helpful to preview the usages to identify those that should be modified by hand or be excluded. Depending on the refactoring operation you choose, you can click the **Preview** button in the refactoring dialog to see a preview of the changes that will occur. When you click Preview, a log window is displayed below the source editor. You can see the 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 **Refactor** button in the Preview log window.

How to Rename a Code Element

In the `.java` files of an application, unambiguous definitions and all references to code elements such as a package, class, interface, method, field, parameter, or variable can be renamed. Use the procedure 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 **Show Usages** button to see the usages in a tree format.

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 element name.
   - In a window such as the Applications window or Structure window, select the element 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 Javadoc for Textual Usages** to extend the operation to comments, the bodies of documentation comments, and to annotations.
   - (Package, type, and members elements only.) Select **Search Text 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.

How to Delete a Code Element

JDeveloper deletes the definition of a class, interface, method, or field after a deletion is confirmed, leaving undeleted code as undefined references. Use the procedure 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 window such as the Applications window or 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** > **Delete**.
   - Press Alt+Delete.

     The project files are searched for usages. The Confirm Delete dialog displays how many usages were found.
3. Click the **Show Usages** button to inspect and resolve the usages in the log window.
4. Select **Preview** to inspect the usages of the deleted file in the log window.
5. Click **OK**.

Refactoring Classes and Interfaces

In `.java` files, primary classes and interfaces can be repurposed, or new ones defined. Use the procedure to move a class or interface.

While developing your Java application you can easily define new classes and interfaces and re-purpose existing ones. For example, you can move a package, class, or interface to a different package. You can optionally generate a preview first, which is 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 an Applications window 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 dialog opens.
3. In the **Move To** field, enter the new package name.

   You can also click the **Search** icon next to the field to navigate to an existing package.
4. Set the depth of the text substitution.
   - **Update References In** - In the drop-down list, choose where the references will be updated.
   - **Search Javadoc for Textual Usages** - Extend the operation to the bodies of documentation.
   - **Search Text Files** - 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.

   You can also click **Show Usages** to open a display panel within the Move dialog to see a list of all current usages in the project.
6. Click **OK**.

   If you selected **Preview**, to avoid all the usages being modified, finish the renaming operation from the Preview log window.

Classes can also be moved in the Applications window by dragging multiple classes from one package to another.

How to Duplicate a Class or Interface

JDeveloper duplicates selected primary classes and interfaces, and adds them to the same package as the original. Use the procedure 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 with the class, for example, `client.frame1`.

4. Click **OK**.

   The new class will be added to the project.

How to Extract an Interface from a Class

When a new interface is derived from selected methods, and static fields are defined in a class, the modified declaration of the class displays it as an implementation of the new interface. Use the procedure to extract an interface.

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 an Applications window 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 want to inspect the usages of the class before committing to the extract operation.
8. Click **OK**.

How to Extract a Superclass

An extracted superclasse based on the selected members of a selected class adds additional levels to the hierarchy of written code. Use the procedure to extract a superclass.

Extracting a superclasses allows you to add additional levels to a hierarchy even after the code is written.You can create a superclass based on chosen members of a selected class. The superclass consists of field and method declarations that match the chosen members.

To extract a superclass:

1. In an Applications window, 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** field, enter the name of the package to which the new superclass will belong.
4. In the **Class Name** field, enter a name for the new superclass.
5. In the **Members to Extract** table, select the members that will be included in the new superclass.

   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 want to convert existing declarations that name the class into declarations naming the superclass.
7. Select Preview to view a list of the usages before committing to their replacement. This option is enabled only if you have selected **Replace Usages**.
8. Click **OK**.

How to Use Supertypes Where Possible

Generalize type specifications of declarations in Java code into supertypes by replacing references to a selected class with references to one of its supertypes. Use the procedure to generalize declarations.

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 an Applications window or in the Structure window, select the name.
2. From the Main menu, choose **Refactor** > **Use Supertype Where Possible**.

   The Use Supertype dialog displays.
3. In the **Supertypes** table, select the supertype that the declarations will be generalized to.
4. Click **OK**.

How to Convert an Anonymous Class to an Inner Class

Unnamed inner classes or anonymous classes can declare and instantiate a class at the same time, and named inner classes can access everything in their enclosing class. Use the procedure to convert an anonymous class into an inner class.

You can declare an inner class within the body of a method. You can also declare an inner class within the body of a method without naming it. This is known as an anonymous inner class. JDeveloper allows you to 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**.

How to Move an Inner Class

Converting an inner class to a top level class renders code less bulky and more readable. Use the procedure to move an inner class.

You can move an inner class to a newly created class at the top level. You might do this because the class is in the wrong package and you want to move it to another package where it fits better.

To move an inner class:

1. Select the inner class name in the Structure window or in a Java source editor window.
2. On the main menu select **Refactor** > **Move**.

   The Move Inner Class dialog displays.
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**.

Refactoring Class Members

Refactoring that involves relocating member definitions from one class to another restructures the code to make it simple and readable.

While developing your Java application, you can easily move member definitions from one class to another. For example, you can move a class member (for example, a method) to another class.

Moving a Static Method

Static methods belong to the class and does not use instance variables of any object of the class in which they are defined. Use the procedure to move a static method.

Methods declared with the `static` keyword as a modifier are called static methods or class methods.

A static method does not use instance variables of any object of the class in which they are defined. The method signature for a `main` method is static, which means that you don't need to create an instance of the class to invoke the `main` method. If you define a method to be static, the compiler displays an error message if you try to access any instance variables.

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.

Moving a Non-static Method

Non-static methods can only be called on an instance of the class itself, and they perform something that depends on the individual characteristic of the class. Use the procedure to move a non-static method.

A non-static method can't be referenced from a static context. The only way to call a non-static method from a static method is to have an instance of the class containing the non-static method. A non-static method requires access to instance-level data in the class, for example, a non-static field.

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.

How to Change a Method to a Static Method

A static method, used for memory management, belongs to the class rather than the object of a class, and can be invoked without creating an instance of a class. Use the procedure to change a method to a static method.

You can assign the `static` modifier to a method. You can also specify what action to take when usages in a non-static context are found while making an element static.

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.

How to Change the Signature of a Method

A method signature comprises the method name and the number and type of its parameters. Use the procedure 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.

How to Pull Members Up into a Superclass

Moving definitions of members from a source class to one of its superclasses retains the functionality of the class member while making code simple and readable. Use the procedure to pull a member 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 an Applications window 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 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 want 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**.

How to Push Members Down into Subclasses

Moving the definitions of members from a class to its immediate subclasses cleans up the class hierarchy. Use the procedure to push a member down into a subrclass.

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 an Applications window 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 want 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 want an abstract definition of the member to be left in the source class.
5. In the **Dependencies** column, select a checkbox to cause all the member's dependencies to be pushed down with the member.
6. Click **OK**.

How to Introduce a Field

Covert Java expressions into named elements. Use the procedure to introduce a field.

While developing your Java application, you can easily convert expressions into named elements. For example, you can 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 display 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 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**.

How to Inline a Method Call

Code compilation saves overheads by inlining or including the body of a method into the body of its callers. Use the procedure 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**.

How to Introduce a Variable

In the source editor, add a new variable declaration to a method for a selected expression. Use the procedure to introduce a member.

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. If more than one data type is valid for the field, choose the **Type** from the drop-down list.

   The menu lists all types that are consistent with the expression. The **Type** field is not shown if only a single type is valid.
4. Modify or replace the suggested name for the variable in the **Name** field.

   You can choose a suggested name 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**.

How to Introduce a Parameter

In the source editor, convert a constant expression in a method body into a new parameter for the method. Use the procedure 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**.
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 is not displayed if only a single type is valid.
4. Modify or replace the suggested name for the variable in the **Name** field.

   You can choose a suggested name 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**.

How to Introduce a Constant

Convert a constant expression into a constant reference so that the constant declaration that is initialized by the expression is added to the class. Use the procedure 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 is not shown if only a single type is valid.
4. Modify or replace the suggested name for the variable in the **Name** field.

   You can choose a suggested name from the drop-down menu.
5. Click **OK**.

How to Extract a Method

Create a new private method from the local variables and parameters of a method that conform to certain restrictions. Use the procedure 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**.
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.

How to Extract a Class

To simplify complex classes, extract a new class by replacing the fields and methods of a class. Use the procedure to extract a class.

You can replace the fields and methods of a class by extracting a new class. All references to the fields are updated to access the new class. Extracting a class enables you to manage classes that have become too complex.

To extract a class:

1. In the source editor, select the class that you wish to extract.
2. From the main menu, choose **Refactor** > **Extract Class**.
3. Enter a name for the new class.
4. Select **Inner Class** to make the new class is an inner class.

   If you do select this option, a top-level class is created.

5. Select whether you want to generate `getter` and `setter` methods for the extracted fields in the new class.
6. Select the methods to extract for the new class.
7. Select **Replace Usages** to replace all usages of the extracted class.

How to Replace a Constructor with a Factory Method

Create a factory method and use it to replace the calls of a constructor to overcome its limitations. Use the procedure to convert a constructor into a factory method.

You can convert a constructor into a factory method. Constructors create an instance of a class. Factory methods are static methods that return an instance of the native class. You can use factory methods for situations in which constructors are too limited.

To convert a constructor into a factory method:

1. Select the constructor name in the Structure window or in a Java Source Editor.
2. From the main menu, select **Refactor** > **Replace Constructor With Factory Method**.

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**.

How to Encapsulate a Field

Encapsulation makes field data accessible only by public methods, thus hiding the private fields and their implementation for outside classes. Use the procedure to encapsulate a field.

Encapsulation makes the fields in a class private and provides access to the fields via public methods. If a field is declared private, it cannot be accessed by anything outline the class. This hides the fields within the class.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**.
3. In the **Fields** table, check the box next to each field that you want to be encapsulated.

   For each field, you can, you can also specify options for method/field accessibilities and the scope for replacements.

4. In the Replace Accessors box, 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. Click **OK**.

How to Invert a Boolean Expression

JDeveloper updates all references to maintain code functionality if a boolean field, parameter or local variable is initialized after its value has been reversed. Use the procedure to invert a boolean method.

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 8-2 contains an example of an inverted boolean expression.

Table 8-2 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;}`

Refactoring XML Schemas

Referring schemas are updated to reflect all changes made to the namespace of a schema target.

When a schema's target namespace changes, all the referring schemas are updated to have the new target namespace.

When you change the base type on a simpletype element that has facets, all facets are not removed. Instead, the facets that are still valid are retained.

9

Building Java Projects

JDeveloper supports several ways to build and compile your projects and applications, including the Make and Rebuild operations, Apache Ant, and Apache Maven.
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, and projects.

Ant supplies a number of built-in tasks that allow you to allowing to compile, assemble, test and run Java applications. You use Ant buildfiles written in XML to build your project. Each buildfile contains one project and at least one (default) target.

Maven support is very similar to Ant support. You can configure WebLogic Maven plug-ins in your POM to deploy a Java project to a WebLogic server.

This chapter includes the following sections:

- About Building Java Projects
- Building with Make and Rebuild Commands
- Understanding Dependency Checking
- Compiling Applications and Projects
- Cleaning Applications and Projects
- Building with Apache Ant
- Building and Running with Apache Maven
- Understanding Continuous Delivery and Continuous Integration

About Building Java Projects

Use Rebuild to compile source files irrespective of changes, or use Make to compiles source files with changes or have dependencies with changes. Build Java projects using Ant or Maven that uses a project object model, a project lifecyle, a dependency management system, and a set of plugins.

JDeveloper supports Rebuild operations that compile source files unconditionally. To compile source files that have changed since they were last compiled, or have dependencies that have changed, use the Make command. For each project you compile, you can configure the Java compiler by setting options in the Project Properties dialog.

Ant provides another way to build applications. You can invoke Ant from JDeveloper's main menu to build targets defined in the current project's project buildfile. Ant is integrated into JDeveloper. You can add or create Ant buildfiles for 14c applications and projects and edit them using the XML Source Editor.

Maven is project management tool that provides a consistent, automated build, test and deployment process for your projects through a project object model (POM), a project lifecyle, a dependency management system, and a set of plugins that are shareable by multiple projects. You can obtain a wide array of project information through Maven including dependency lists and unit test reports.

Building with Make and Rebuild Commands

The tabulated Make and Rebuild commands execute standard operations to compile projects in JDeveloper.

The Make and Rebuild commands shown in Table 9-1 execute standard operations for compiling projects in JDeveloper.

Table 9-1 Make and Rebuild Commands

Command	Description
`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.
`Make Project Working Set`	Makes all the projects in the working set.
`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.
`Rebuild Project Working Set`	Rebuilds all the projects in the working set.
`Make All`	Makes all the projects.
`Clean All`	Cleans all the projects.
`Clean Project`	Cleans the project the project depends on (recursively), and then makes the project.

How to Set Compiler Preferences

Set the options for the way the compiler builds code using the Compiler page of the Preferences dialog. Use the procedure to configure the deployment preferences.

You can set compiler options in the Compiler page of the Preferences dialog.

To configure the deployment preferences:

1. Choose **Tools > Preferences** from the main menu.
2. Select the **Compiler** node. Configure the compile options as required. For more information, click **Help**.
3. Click **OK**.

Compiling with Make

Invoke a Make operation on individual source files, on working sets, or on containers to compile source files that have changes or have dependencies with changes. Use a step from the options to initiate a Make operation.

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 want 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 prints on the top row of the Compiler Log window.

To make a source file, do one of the following:

- Right-click in a file's source editor and choose **Make**.
- Select one or more projects in the Applications window, and click **Make** in the toolbar.
- Select one or more projects in the Applications window, and choose a **Make** item from the **Build** menu.
- Select one or more projects in the Applications window, right-click, and choose **Make**.

Compiling with Rebuild

Rebuild operations are not conditional, and compile all the source files in a project or workspace. Use a step from the options to initiate a Rebuild operation.

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 want to compile more selectively, you can add an Ant buildfile to a project, define additional targets, and run Ant to make those targets. Ant supplies a number of built-in tasks allowing you to compile, assemble, test and run Java applications.

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 is printed to the top row of the Compiler Log window.

To rebuild source files, do one of the following:

- Select one or more source files in the Applications window, right-click, and click **Rebuild** (for one file), or **Rebuild Selected** (for multiple files).
- Select one or more projects or workspaces in the Applications window, and click **Rebuild** in the toolbar.
- Select one or more projects or workspaces in the Applications window, and choose a **Rebuild** item from the **Build** menu.
- Select one or more projects or workspaces in the Applications window, right-click, and choose **Rebuild**.

Understanding Dependency Checking

JDeveloper analyzes dependencies while building code. A Make operation for compilation includes dependency checking which accelerates the edit and compile cycle by circumventing needless compiles of interdependent source files.

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>
```

Compiling Applications and Projects

Set the compiler options for the Java Compiler, Javac that JDeveloper uses by selecting Options for Compiler from Project Properties. Javac transforms Java source code into Java bytecode for JVMs.

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. See Understanding Dependency Checking.

When you work inside JDeveloper, the compiler used is Javac. You can adjust compiler options by choosing **Project Properties** > **Compiler** > **Options**. Each option on the Compiler: Options page contains a description alongside it.

How to Configure Your Project for Compiling

Set the options in the Project Properties for how the Java compiler builds code. Use the procedure to configure project properties 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 to display compiler messages such as:

- `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 Applications window and choose **Project Properties** from the context menu.

   You can also double-click a project node in the Applications window.
2. In the Project Properties dialog, expand the **Compiler** node.
3. Click **Options**.
4. Under **Compiler Options**, expand the **Javac** node.
5. 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 do not want to display the `-Xlint` message, go to **Compiler** > **Options** and expand the **Javac** > **Warnings** node in the Project Properties dialog. This allows you to specify which Xlint messages are displayed.

6. Optionally expand the **Turn Individual Warnings Off** node.
7. Uncheck the **-Xlint Unchecked** checkbox.
8. Close all dialogs and recompile.

> **Note:**
>
> If you want to have all your project files automatically saved before compiling, specify this in the Compiler page of the Preferences dialog.

How to Specify a Native Encoding for Compiling

The compiler uses the default native-encoding converter for the platform if no encoding scheme to control how the compiler interprets multibyte characters is set. Use the procedure to set an encoding option.

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. In JDeveloper, select **Project** > **Project Properties**.
2. In the Project Properties dialog, select the **Compiler** node.
3. On the command line, use the javac command with the `-encoding` option followed by the encoding name.
4. 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.

Compiling from the Command Line

Compile applications and projects from within JDeveloper using the Build and Compile options, or compile from the command line by using the `ojmake` and `ojdeploy` tools.

There are 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 `ojmake` and `ojdeploy` 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`.

For information about using ojmake, see Using ojdeploy and ojmake.

For information about using ojdeploy, see Using ojdeploy.

Cleaning Applications and Projects

Specify the output location of the project to be cleaned as a directory, and use the Clean command to remove class files, all copied resource files, and all deployed files from the output, and deploy directories in a project or application. The Clean All and Refresh Application command also cleans IDE artifacts such as index files and deploy files.

Cleaning enables you to remove previous build artifacts and start afresh. 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.

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.

The Clean All and Refresh Application command performs all the functions described above but it also cleans IDE artifacts such as index files, deploy files, etc. If these artifacts are not cleaned, you may encounter errors during compilation or at runtime because these IDE artifacts may be out of sync with the current state of the application.

How to Clean

The Clean command enables you to remove artifacts left over from previous builds in order to begin a fresh build process.

Cleaning a Project

Cleaning a project prevents errors during compilation or at runtime. Use the procedure to clean a project.

You can clean a single project within an application.

To clean a project:

1. In the Applications window, select the project to be cleaned.
2. In the **Build** menu, select **Clean <*project*>**.
3. In the Cleaning <*project*> dialog, click **Yes**.

Cleaning an Application

Cleaning an application cleans the application and all of its projects. Use the procedure to clean an application.

Cleaning an application cleans the application and all of its 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.

To clean an application:

1. In the Applications window, select the application you want to clean.
2. In the **Build** menu, select **Clean All**.
3. In the Cleaning *application* dialog, click **Yes**.

Cleaning and Refreshing an Application

Select the Clean All and Refresh Application option to sync the IDE artifacts with the current state of the application and clean IDE artifacts to prevent errors at compilation or runtime. Use the procedure to clean and refresh an application.

You can clean and refresh and application with a single command.

To clean and refresh an application:

1. In the Applications window, select the application to be cleaned and refreshed.

2. In the **Build** menu, select **Clean All and Refresh Application**.
3. In the Cleaning All and Refresh Application dialog, click **Yes**.

Building with Apache Ant

Ant, a Java library and command-line tool written in Java, is integrated with and installed with JDeveloper. Ant is extensible, supports cross-platform compilation, and uses XML formatted buildfiles to describe and control build processes for applications and application components.

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. See `http://ant.apache.org/index.html`.

An Ant buildfile defines targets and dependencies between targets. Each buildfile contains one project and at least one target. A target is a sequence of programmatic tasks. When you run Ant to make a target, it first makes other targets on which it depends, and then executes the target's own tasks.

Ant is integrated and installed as part of JDeveloper, which means you do not need to add it as an extension. You can add or create Ant buildfiles for applications or for projects, or you can create and empty buildfile. You can use the XML Source Editor in JDeveloper to edit Ant buildfiles.

Create an Ant Build File at Application Level

An Ant `buildfile` defines targets and dependencies between targets. Use the procedure to create an Ant build file at the application level.

To create an Ant build file at application level:

1. Open the New Gallery by choosing **File > New**.
2. In the New Gallery, in the **Categories** tree, under **General**, select **Ant**.
3. In the **Items** list, double-click **Buildfile from Application**. The Create Buildfile dialog opens where you can change the default filename and directory. For help with the dialog, press F1.
4. Click **OK**. A new `build.xml` file is created and opened in the XML Source Editor. The generated `build.properties` file is also created. Both files are listed in Application Resources in the Applications window.

Create an Ant Build File at Project Level

Use the procedure to create an Ant build file at the project level.

To create an Ant build file at project level:

1. Open the New Gallery by choosing **File > New**.
2. In the New Gallery, in the **Categories** tree, under **General**, select **Ant**.
3. In the **Items** list, double-click **Buildfile from Project**. The Create Buildfile dialog opens where you can change the default filename and directory. For help with the dialog, press F1.

4. Click **OK**. A new `build.xml` file is created and opened in the XML Source Editor. The generated `build.properties` file is also created. Both files are listed in Application Resources in the Applications window.

Create an Empty Ant Build File

Use the procedure to create an empty Ant build file.

To create an empty Ant build file:

1. Open the New Gallery by choosing **File > New**.
2. In the New Gallery, in the **Categories** tree, under **General**, select **Ant**.
3. In the **Items** list, double-click **Empty Buildfile**. The Create Buildfile dialog opens where you can change the default filename and directory. For help with the dialog, press F1.
4. Click **OK**. A new `build.xml` file is created and opened in the XML Source Editor, and it is listed in the Applications window under the Resources node.

Example 9-1 Running Ant on buildfile targets:

- On targets in the project buildfile. A project can contain several Ant buildfiles, but only 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 window when editing an Ant buildfile. When an Ant buildfile is open in an XML source editor, its targets are listed in the Structure window. 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.

Running Ant on Project Buildfile Targets

A project contains one designated project `buildfile` to build tasks that work towards a required state during the build process. Invoke Ant in JDeveloper to build targets defined in the project `buildfile`.

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 buildfile, go to the Ant project properties page (choose **Project** > **Project Properties** > **Ant**).

You can 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 Structure Pane, choose a target.

Using the Ant Tool in the IDE

The Ant Log window displays messages pertinent to an Ant build, such as building targets, compilation errors, and warnings pointing to the source code, and the output level of the messages.

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.
- 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.

Building and Running with Apache Maven

Apache Maven is a build process automation tool that manages a project build, reporting, and documentation from the project object model. Use Maven to build projects, manage dependencies on Oracle and third party artifacts, and to download artifacts.

Apache Maven is a widely used tool for managing and automating the build process. It is also used for project management, in particular dependency and release management.

It provides consistent and automated build, test and deployment of applications built using JDeveloper. One advantage of using it is that a developer who has worked on a project that uses Maven is usually able to transfer to another project that uses Maven with very little effort. Most developers who have used Maven before are able to get an unfamiliar project that uses Maven downloaded, built and deployed quickly.

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. A Maven lifecycle consists of the processes for building and processing an artifact, that is a project. The following are different types of Maven build lifecycles:

- Default - project deployment
- Clean - project cleaning
- Site - project site documentation

Each build lifecycle is defined by a set of build phases where each phase a stage of the lifecycle. For example, a default build lifecycle can consist of the following phases:

- validate - verify the project is correct and all necessary information is available
- compile - compile the source code of the project
- test - test the compiled source code using a suitable unit testing framework
- package - take the compiled code and package it in its distributable format, such as a JAR.
- install - install the package into the local repository, for use as a dependency in other projects locally
- deploy - copy the final package to the remote repository for sharing with other developers and projects

In turn, each phase is defined by a set of plugin goals (tasks) that determine specifically how the phase is accomplished. Goals and phases are run sequentially and in the order they are invoked. JDeveloper enables you to define the phases and their goals and invoke them from the POM file context menu. For more information, see Using the Context Menu in the POM file .

If you are developing an application using Oracle Fusion Middleware, you can use Maven to:

- Build projects.
- Manage dependencies on Oracle and third party artifacts.
- Download artifacts automatically from an internal or external (public) Maven repository.

For information about Maven, see `http://maven.apache.org/index.html`.

Understanding Repositories

A Maven **repository** is sharable location that hosts a collection of artifacts. Artifacts can be pulled in from central public repository or private repositories. Maven updates artifacts from specified public or private repositories. Each artifact is pulled into your local Maven repository as a build executes. The artifacts are organized in a particular directory structure. These include compiled code (JAR files, WAR files) and metadata about that code.

There are two types of repositories:

- Local Repository - A cache of a remote repository that is stored on the local machine. Maven projects are built against the local repository. The local repository usually only stores a subset of the files available in the remote repository and any temporary build artifacts. Maven accesses this location for resolving any artifact. You should not edit the contents in this location.

  When you execute a Maven goal to build a project, Maven downloads any necessary dependencies from the upstream repositories and saves them in your local repository. Maven accesses this location for resolving any artifact.

- Remote Repository - A repository that contains all the Maven artifacts and plugins. The remote repository may be a third-party repository (for example, http://repo.maven.apache.org/), or it may be a private internal repository.

JDeveloper manages Maven repositories and makes them available to you, typically through HTTP. It may also proxy (cache artifacts from) other, usually external, Maven repositories to shorten build times and reduce network usage. This means that a Maven POM is provided for each developer-focused product artifact. You are then able to configure Maven to point to this Maven repository.

Use the Maven Synchronization tool to facilitate setting up and populating the repositories. The synchronization tool is a plugin available in the Oracle JDeveloper and the Oracle WebLogic Server installations. To add this plugin, see Populating the Repository.

Understanding Maven Plugins

Extend Maven with plugins because Maven is a plugin execution framework whose work is done by plugins. Configure core plugins such as build plugins that run during the build, and reporting plugins using the <build/> and <reporting/> elements from the POM.

You can extend Maven with plugins. These provide a number of other development tools for reporting or the build process. Plugins allow running Maven goals. All work is done by plugins. There are a number of core plugins:

- Build plugins execute during the build and should be configured in the `<build/>` element from the POM.
- Reporting plugins execute during the site generation and should be configured in the `<reporting/>` element from the POM.

For a list of available Maven plug-ins, see `http://maven.apache.org/plugins/index.html`.

Understanding Dependencies

Maven dependencies specify the libraries containing the jar files required for project builds. Compilation causes Maven to download and link dependencies, and all dependencies in the POM file are always kept in sync with the `.jpr` project or the `.jws` application.

You use a Maven dependency to specify a library containing the jar files required for building your project. If you build a project with a dependency that does not exist in a local repository, Maven will search for it and add it to your local repository. If the project is a Maven project (that is, has an associated POM file) and dependencies are added to the project, the changes are committed to the POM as well.

Maven downloads and links the dependencies for you on compilation and other goals that require them. It also brings in the dependencies of those dependencies (transitive dependencies), allowing your list to focus solely on the dependencies your project requires.

Dependencies are synchronized between the selections on the Dependencies tab and the POM file. For example, if a project has a dependency that is not in the POM and you add another dependency from the project, both dependencies are added to the POM when you click **OK** on the Maven: Dependencies dialog.

The dependencies in the POM file are always kept in sync with the `.jpr` project or the `.jws` application associated with the POM file. The dependencies (except from the repository) can also be added from project properties and application properties dialogs, they are automatically synced to the associated POM file.

Understanding the Project Object Model

Project Object Model represents a Maven project held in a file named `pom.xml` that contains project information and the plugins configurations required for the build process. The required fields, `groupId:artifactId:version` and `packaging` point to a specific project version and behave like an address and timestamp.

A Maven project object model (POM) file is similar to a JDeveloper `.jpr` file. The 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.

Maven uses the concept of "convention over configuration" by assuming a standard default behavior for projects. For instance, for a Java application, Maven assumes the location of source code is `${`*basedir*`}/src/main/java`, resource files are assumed to be in `.../main/resources`, and so on. Following convention then, Maven assumes that ultimately a JAR is to be created in a directory `${`*basedir*`}/target`. All of this information is already contained in the POM, relieving you of having to configure each and every path. The Maven core plugins also employ conventions for compiling source code, packaging, and other processes. By following convention, Maven can perform most of the work of building and managing a project for you.

The following are the minimal Maven coordinates that a POM can contain:

- **Group ID** - a unique identifier for the project that is similar to a package name, for example, `com.acme.corp`.
- **Artifact ID** - the name of the JAR without the extension, for example, `MyProject`.
- **Version** - the current version of the artifact produced by this project, for example `1.1.0`.

Packaging is another coordinate that you can specify when creating a POM. You use packaging to specify the project's artifact type (for example, you specify that the project is packaged as a JAR or a WAR file). If you do not specify a value for packaging, Maven assumes the default type is a JAR.

For information about the Maven Project Object Model, see: `http://maven.apache.org/index.html`

Understanding the Settings File

Use the main Maven control file, the `settings.xml` file, that is common to all projects, to configure Maven execution, such as setting the local repository path, setting an alternate remote repository servers, and setting proxies. Use the procedure to configure the `settings.xml` file.

The `settings.xml` file is the main Maven control file. You use this file to configure Maven execution such as setting the local repository path, setting an alternate remote repository servers, setting proxies, and more. However, unlike the POM file, the `settings.xml` file should remain common to all projects (recommended) until you explicitly change it.

To configure the settings.xml file:

1. In the main menu, choose **Tools** > **Preferences**.
2. Select **Maven** > **Settings** in the category tree.
3. Check the command line options you want.

For information about the settings file, see `http://maven.apache.org/settings.html`.

Selecting the POM File

Select a POM for an application and each project so as to sync the paths and the dependencies of the `.jpr` file. Use the procedure to set the application POM file.

When you use Maven to create and manage your application, a POM is created at the application level and for each project within the application. You can specify the POM files to use for the application and each project. When you select the POM file for a project, the paths and the dependencies of the `.jpr` will automatically be in sync.

> **Note:**
>
> Typically you do not need to explicitly set the POM files as they are set when you create the application or project when you select Maven as the build tool in the application or project creation wizard. The POM files are also set if you choose to create a POM for your application or project from the gallery.

To set the application POM file:

1. Right-click the application in the Applications Window and choose **Application Properties**.

2. Select **Maven** in the category tree.

   If this is the first time you have selected Maven for this application, click **Load Extension**.

3. Click the **Browse** button and select the POM file you want to use.

To set the project POM file:

1. Right-click the project in the Projects window and choose **Project Properties**.
2. Choose **Maven** in the category tree.
3. Click the **Browse** button and select the POM file you want to use.

Installing Maven

Maven is provided by JDeveloper and can be found in the *ORACLE_HOME*/oracle_common/modules directory. If you want to install Maven on a server, see Installing and Configuring the Archiva Maven Repository Manager.

Before You Begin

Use the procedure to create a Maven application and project for use. Create an application and generate POM files for an application and all of its projects using the New Gallery.

To use Maven, you first create an application and then select options for at least one of the projects within it.

You can generate POM files for an application and all of its projects using the New Gallery. 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.

To create a Maven application and project for use:

1. From the Main menu, select **File** > **New** > **From Gallery** > **Maven** > **Application POM**.
2. Follow the instructions in the dialog to specify options that create an application that will be configured for use with Maven technologies.
3. Ensure that you enter the options marked **(required)** in the wizard:
   - **Group ID**
   - **Artifact ID**
   - **Version**
4. When you complete the wizard, the application displays in the Applications window with a single Maven project under it.

   > **Note:**
   >
   > You can also create another type of application, for example, a Custom application, and import an existing Maven project to it using the New Gallery.

5. In the Applications window, expand a project node and it's Resources node.
6. Double-click the POM file to open it.

   By default, the name of the POM file is pom.xml. It is usually located in the Applications window under Projects > Resources.

7. In the Overview tab, click **Repositories**.
8. In the **Local Repositories** field, check for a red underline.

   A red underline indicates that the directory being pointed to is an invalid one. It does not show up for a valid directory.

   This field displays the location of your local Maven repository. The repository holds a local copy of all artifacts and dependencies that your project may need.

   By default, the location of the local repository is `USER_HOME/.m2/repository`. If the location of the local repository is underlined in red, the location is invalid. Create a directory using file explorer or using the browse icon next to the **Local Repositories** field. Once you select a valid directory, the red underline disappears.

   The local repository remains common to all projects (recommended) until you explicitly change it.
9. Choose **File** > **Save** to save any changes.
10. From the Main Menu, choose **Tools** > **Preferences** > **Maven** > **Settings**.
11. Optionally change the path in the **User Settings** field.

    When you create the Maven application, a file called `settings.xml` is also created in `USER_HOME/.m2/` by default.
12. Click **OK**.
13. From the Main menu, select **Tools** > **Preferences** > **Web Browser and Proxy**.

    By default, `settings.xml` does not contain active proxy information, but JDeveloper has proxy settings enabled.
14. Click the **Proxy Settings** tab and select either **Use System Default Proxy Settings** or **Manual Proxy Settings**.

    Maven uses the proxy to download artifacts from remote repositories.
15. Click **OK** when you are finished.

How to Create Maven POM Files

You can add a POM file to a project that is created in a Maven-enabled application. Use the procedure to create a POM.

If you create a project in a Maven-enabled application, you can add a POM file to the project.

To create a POM:

1. In the Applications Window, select the project or file where you want to locate the POM.
2. Choose **File** > **New** > **From Gallery** to open the New Gallery.
3. In the **Categories** list, expand **General** and select **Maven**.
4. In the Items list, select **Maven POM for Project.**
5. Click **OK**.
6. Change any required and optional options in the dialog as needed.
7. Click **OK**.

   The POM file is created under the Resource node of the project and the file is opened in the Overview editor by default.

Using the Context Menu in the POM file

Specify the phases or goals to include in a lifecycle phase to create and manage a goal profile. Use the procedures to create or manage a goal profile, and to add a goal.

You can access a menu of commands such as **Run Maven Goal Profile** profile and **Run Maven Phase** phase by right-clicking the `pom.xml` file in the Applications window or the `pom.xml` Overview or Source tab if you have the POM file open.

You can create and manage a goal profile by specifying the phases or goals to include in a lifecycle phase:

To create or manage a goal profile:

1. Right-click a POM file.
2. Select **Manage Goal Profiles**.
3. In the Maven: Phases/Goals preference page, select the goal profile you want (or click the plus icon to create a new profile).
4. Select the lifecycle phases you want and click the right arrow to move them into the Selected Phases/Goals column.

   Hold down the Shift key to select and move multiple phases at once.
5. Click **OK**.

You can customize the list of phases and goals by adding or deleting goals from the Selected Phases/Goals list. For instance, if you want to be able to undeploy a Maven application, you can add the undeploy goal from the `weblogic-maven-plugin` available from the local repository.

To add a goal:

1. Click the plus (+) sign in the Selected Phases/Goals panel.
2. Locate and expand the plugin node containing the specific goals you want to add.
3. Select the goals (press Shift down to select multiple goals in sequence or press Ctrl to select non-adjacent goals) you want.
4. Click **OK**.

If the goal you want is not available in the local repository, click Download from Remote Plugin Repository and specify the name of the goal, the plugin version, and select the remote repository containing the plugin. To ensure you have access to the remote repository, click **Verify Plugin**.

For information on running Maven goals from the command line, see "Running the Oracle Maven Synchronization Plug-In".

For information on Maven phases and goals, see `http://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html`.

How to Specify and Manage Remote Repositories

If a project needs artifacts in remote repositories, use the Tools menu or the POM file of the project to manage the remote repositories. Use the procedures to add and manage remote repositories by means of Preferences in Tools, and by means of the POM file.

You need access to remote repositories for a number of reasons. For example, your project might use dependencies from different external vendors. These dependencies may be available in different remote repositories. JDeveloper allows you to configure and search for artifacts in remote repositories.

You can manage remote repositories through **Tools** > **Preferences** from the main menu or through a project's POM file. If you add a remote repository through the Preferences dialog, you can add the repository to the `settings.xml` file making it available to all projects in the application. If you add the remote repository to the project's POM file, it is available only to that project.

To add and manage remote repositories through Tools > Preferences:

1. From the Main menu, click **Tools** > **Preferences** > **Maven** > **Repositories**.
2. In the Remote Repositories table, click the **Add** (+) icon.

   A new row is added to the table.
3. Select the new row.
4. In the Repository Details section of the Maven - Repositories page, enter a **URL** to the remote repository, for example, `http://repo1.maven.org./maven2`.
5. In the **Index Update URL** field, enter the base location from where `nexus-maven-repository-index.zip` and `nexus-maven-repository-index.properties` will be downloaded from.

   You must provide the index location to search for artifacts in the repository without having to actually download them.

   > **Note:**
   >
   > You can check the accessibility to the repository and its index data by clicking the **Test** button.
6. Select **Include in Settings.XML** to add the repository to the settings.xml file.
7. Select the checkbox for the new repository in the Remote Repositories table.
8. Click the **Index Selected Repositories** icon (located to the right of the **Delete Repository** icon).

   This downloads the indexing data for the selected remote repository.

To add and manage remote repositories through the POM file:

1. Double click the `pom.xml` file for your Maven project in the Applications window. Details of the POM file are displayed in the IDE. By default, the General tab is opened.
2. Click the Repositories tab.
3. In the Remote Repositories table, click the **Add** (+) icon.

A new row is added to the table. Alternatively, you can open the drop-down list next the Add icon and select **Add Repository** to add a new row.

Alternatively, you can choose **Add From Preferences** to add a remote repository from a list of remote repositories. Select the desired repository and click **OK** to add it to the list of remote repositories.

4. Enter the name, ID, and URL of the new repository.
5. Check **Repositories** to include this repository in the Repositories section of the POM file.
6. Check **Plugin Repositories** to include this repository in the Plugin Repositories section of the POM file.

   If you are entering a new remote repository, you can click **Add to Preferences** to add the selected repository to the Preferences list.

Once added, you can specify the following details for the selected repository:

- Layout - Whether the directory structure of the repository is legacy (Maven 1) or the default (Maven 2). The layouts differ in directory structure, timestamp of snapshots in default, and existence of metadata files in default.
- Do Not Search for Releases - Whether Maven should look for release versions.
- Do Not Search for Snapshots - Whether to depend on snapshot releases. Note that is not advisable to release or deploy an artifact that has a dependency on a snapshot version of an artifact.
- Update Policy - How Maven should look for updates in the repository (always, daily, interval, or never) when searching for release version and for searching for snapshots.
- Checksum Policy - Which checksum policy to use (warn, fail, or ignore) when searching for release version and for searching for snapshots.

Populating the Repository

Upon a Maven project creation, JDeveloper downloads the latest versions of the artifacts into the local repository, and the Maven Synchronization plugin available with JDeveloper populates the local or shared repository with Maven artifacts. Use the procedure to run the Maven Synchronization plug-in in a shared repository.

When you create a Maven project, the IDE downloads the most recent artifacts into your local repository. In a typical development scenario, these dependencies are downloaded from a shared repository, although JDeveloper can also populate your local repository from IDE library definitions.

Oracle recommends development teams maintain a shared repository containing the artifacts the application depends on for compilation. To populate a shared repository containing JDeveloper and ADF libraries for teams to develop against, you can use the Maven Synchronization plugin (also referred to as the sync tool).

The Maven Synchronization plugin populates your local or shared repository from your *ORACLE_HOME* directory. The plugin is available with JDeveloper. The Synchronization plug-in consists of two components:

- `oracle-maven-sync-`*`version`*`.pom` - Describes the plug-in. The value of *`version`* is the version of the Maven sync POM file, for example, `oracle-maven-sync.12.1.3.0.0.pom`.
- `oracle-maven-sync-`*`version`*`.jar` - Contains the plug-in. The value of *`version`* is the version of the Maven sync JAR file, for example, `oracle-maven-sync.12.1.3.0.0.jar`.

Both components are located at *ORACLE_HOME*/oracle_common/plugins/maven/com/oracle/maven/oracle-maven-sync/*version*.

The synchronization plug-in checks for all Maven artifacts in *ORACLE_HOME*, ensures that all artifacts are installed in the specified repository, and that the versions are an exact match. This means that the version numbers and the files are exactly the same at the binary level, ensuring that all patched files reflect accurately in the Maven repository, which is essential for a successful deployment.

Maven artifacts like plugins are installed in ORACLE_HOME/maven/artifact.

To run the Maven Synchronization plug-in in a shared repository:

1. Install the Maven Synchronization plug-in:

   ```
   mvn install:install-file -DpomFile=/oracle-maven-sync.version.pom -
   Dfile=oracle-maven-sync.version.jar
   ```

2. Run the sync tool to populate the shared repository:

   ```
   mvn com.oracle.maven:oracle-maven-sync:push -Doracle-maven-
   sync.oracleHome=ORACLE_HOME -Dmaven.repo.local=/alt_path
   ```

> **Note:**
>
> For initial testing and prototyping of your Maven project, JDeveloper will populate a local repository for you, however, this is not recommended.

For information on installing and running the Maven Synchronization plug-in, see "Populating the Maven Repository Manager".

Once you have populated the repository, you have access to many archetypes, including the ADF `oracle-adffaces-ejb` archetype, which provides the basic template for Fusion Web applications using the Model and ViewController projects. When you use this archetype to create an application, three POM files are generated:

- A POM for the EJB project
- A POM for the WAR (JSF) project
- A POM that defines the EAR that packages the EJB and WAR projects

The project POM files contain the `ojmake` plugin and `ojdeploy` plugin. Both project POMs point to the parent POM that refers to a parent project from which the other projects inherit dependencies, plugins, plugin configurations, repositories, and more.

For information on creating an application using a Maven archetype, see How to Create Maven Projects Using Maven Archetypes.

Synchronizing POM and Project Files

When the project source path or the classpath for Maven-enabled projects is updated, the change to the POM file is synchronized in JDeveloper. If a dependency is added in the POM source file for a project, JDeveloper adds that artifact to the project `.jpr` file.

For Maven-enabled projects, when you update a project source path or classpath, JDeveloper synchronizes the change to the POM file. For example, if you go to the Libraries and Classpath panel of the Project Properties dialog and add a JAR file, JDeveloper creates a POM for that

JAR file, loads it into your local repository, and adds a `<dependency>` element in the project's POM file. Similarly, if you add a dependency in the POM source file for your project, JDeveloper adds that artifact to the project file (`.jpr`).

Customizing Maven Synchronization

Upon a POM file modification, Maven automatically updates the project files to keep the project and POM in sync, and conversely, whenever a project is modified, the POM is automatically synced. Use the procedure to switch to manual synchronization.

Each time a POM file is modified, Maven automatically updates the project files to keep the project and POM in sync. The reverse is also true, that is whenever a project is modified, the POM is automatically synced.

Automatic synchronization between the project and POM is a setting in the application properties Maven page and is set to **ON** by default. In some cases, however, you might prefer to control when synchronization occurs, for instance:

- You are making several changes to a project or POM and you want to sync the repository once after all changes have been made instead of having syncing occurring at each change
- You have updated the project but do not want the POM to be updated immediately
- Your project is under source control and after making changes, you discover that the POM file or some project files were not checked causing synchronization to fail and you want to be able to manually trigger synchronization after checking out all files
- Libraries or classpath dependencies added to the project are not present in any of the project's repositories, and you want to manually trigger the deployment of libraries and classpath dependencies to the local repository

You can switch the synchronization setting to manual mode and trigger syncing to occur (from POM to project or project to POM) when you want it. You can also set an audit rule to alert you if the project and POM become out of sync when in manual mode. You can then elect to run a sync operation from the audit window. For information on setting an audit rule, see 11.2.7 How to Audit Java Code in JDeveloper.

To switch to manual synchronization:

1. Right-click an application and choose Properties from the context menu.
2. Select **Maven** in the Properties window.
3. Select **Manually manage POM-Project synchronization**.
4. Deselect **Automatic synchronization**.
5. Click **OK**.

When you are ready to sync the POM and project, go back to the Maven page in the Properties window and click **Sync Now**. Depending on what was modified, you can sync the POM to the project or the project to the POM.

Some project dependencies have dependencies of their own, referred to as transitive dependencies. When syncing dependencies from the POM to the project, any transitive dependencies are also synced at the same time.

How to Match the Default Maven Structure When You Create an Application

When a new Maven application is created, the normal project structure can be modified to match the default Maven structure. Use the procedure to modify the default Maven structure.

You can match the default Maven structure when you create a new Maven application.

To modify the default Maven structure:

1. Choose **File** > **New** > **From Gallery** to open the New Gallery.
2. In the **Categories** list, expand **General** and select **Applications.**
3. Choose **Custom Application**, then **OK**.
4. Choose default options and click **Next**.
5. On the **Project Features** tab, scroll down to **Maven**.
6. Click the shuffle icon to move **Maven** to the **Selected** column of the **Project Features** tab.
7. Click **Next**.
8. Select the **Modify Normal Project Structure To Match The Default Maven Structure** checkbox.

   This is selected by default.
9. Click **Finish** to create the custom application.

How to Create Maven Projects Using Maven Archetypes

Maven Archetype is a project template toolkit from which the src-main and src-test types of source code are generated for a project. Use the procedure to create a Maven-enabled J2EE application using an archetype.

Archetypes are similar to templates that you can use to create applications quickly using Maven. The archetype provides a consistent means of generating Maven projects. For more information, go to `http://maven.apache.org/guides/` and see the links for "What is an Archetype" and "Creating Archetypes."

To create a Maven-enabled J2EE application using an archetype:

1. Choose **File** > **New** > **From Gallery**.
2. In the **Categories** list, expand **General** and select **Maven** > **Generate from Archetype**.
3. In the Create Project from Archetype dialog's Select an Archetype page, enter options for the new project. Ensure that you enter the options marked **(required)** in the dialog.
4. Click the browse icon next to the **Maven Archetype** field.
5. In the search field in the Search for Archetype dialog, enter search text and any filters to find the archetype you want to use.

   For example, you could enter `j2ee` as your search text to find an archetype called `maven-archetype-j2ee-simple`. You can also choose to narrow the search to a particular repository or repositories.

> **Note:**
>
> The list of available repositories is set in the Maven - Repositories page.
>
> To access a remote repository, make sure you have specified proxy information on the Web Browser and Proxy dialog.

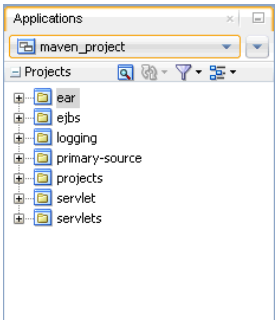
6. Select the repositories to search in the Repositories panel.
7. Press the Return key.

   The **Matching Archetypes** list populates with all of the archetypes that meet your search criteria.
8. Drill down to an archetype in the list and select it, for example, **maven-archetype-j2ee-simple**.
9. Click **OK**.

   The archetype source is displayed in the Maven Archetype field on the Select an Archetype page.
10. Click **Next**.
11. Change the value of a parameter if needed by double-clicking in the Parameter Value column for the selected parameter, and enter a new value.
12. Click **Finish**.

    A new Maven application is created from the archetype. It displays in the Applications window, as shown in Figure 9-1.

Figure 9-1 Maven Application in Applications window



When you create a maven project using an archetype, two types of source code are generated. A `src/main` and `src/test` are generated. Project source code and resources are placed under `src/main` while test cases for the project are placed under `src/test`. You can find Java classes such as JUnit or TestNG tests in this directory along with classpath resources for tests.

> **Note:**
>
> You can also import existing Maven projects into JDeveloper by selection using **New Gallery** > **General** > **Maven** > **Import Maven Projects**. This creates a Maven application and associated projects based on the contents of the imported Maven source.

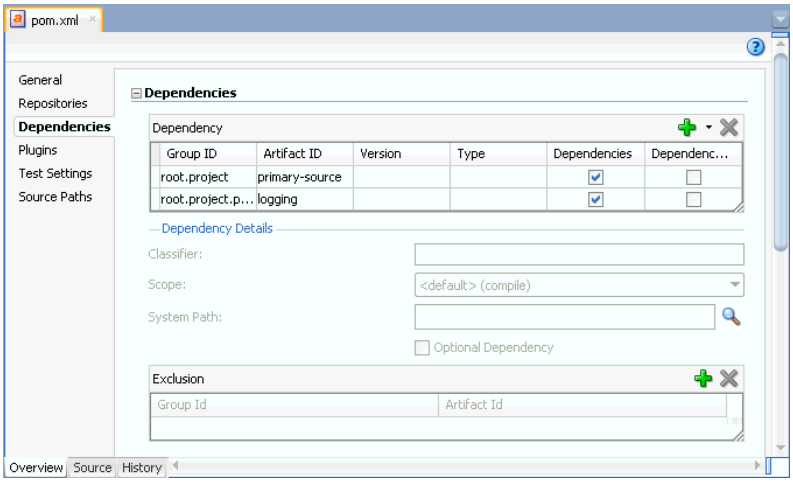
What Happens When You Create a New Maven Application

Use the procedure to view dependencies for a Maven application in Dependencies for **ejbs** Resources.

The application in Figure 9-1 contains a number of Maven projects.

1. In the Applications window, scroll to the **ejbs** project under the Maven application you just created.
2. Expand **ejbs**, then **Resources**.
3. Double-click the project file. By default this should be **pom.xml**. The Editor opens.
4. Click the **Dependencies** tab.
5. Notice the dependencies on the Dependencies tab, for example, those shown in Figure 9-2:

Figure 9-2 Maven Dependencies



6. From the Main toolbar, choose **Project** > **Project Properties** > **Libraries and Classpath**.
7. Notice that the same two dependencies in Figure 9-2 are listed here.

   The dependencies are kept in sync between the application/project and its associated project POM files. Try adding new dependencies in the application/project or POM file to observe the sync process. The sync is required to manage the application/project using both JDeveloper build tools and Maven.
8. In the Project Properties dialog, click on **Maven**.

   This POM file is set as default POM for the project.

How to Run Maven Goals on POM Files

Maven plugin goals, such as clean, compile, build, and package, represent stages that comprise build phases in build lifecycles. Use the procedure to run Maven goals.

Maven is based around the concept of a build lifecycle. A build lifecycle is made up of build phases, wherein a phase is comprised of plugin goals that represent a stage in the lifecycle. Maven provides some default goals such as clean, compile, build, and package to manage your project. Every project runs the same core phases which includes the goals in the phase.

To run Maven goals:

1. From the Main menu, select **Tools** > **Preferences** > **Maven** > **Phases/Goals**.

   You can also right click on a POM file in the Overview or Source tab and choose **Manage Goal Profiles**.

2. To optionally set up a profile, a collection of goals that you can save for future use, click the plus (+) icon next to the **Goal Profile** field.

   - In the Goal Profile field, enter a name for your profile and click **OK**.

3. In the **Available Lifecycle Phases** list, select one or more goals.

> **Note:**
>
> The first time you try this, make sure the default set of goals are included in this list.

4. Click the shuttle button to add the selection to the **Selected Phases / Goals** list.
5. Click **OK**.
6. In the Applications window, scroll down to the POM file for the project you want to run.
7. Right click and **Run Maven Phase "compile"**.

   This downloads some artifacts into the local repository from a remote central repository automatically that are required to execute the compile goal. This is one-time process. Once you have a copy of those artifacts in your local repository, Maven no longer needs to download them again.

   After downloading, the compile goal executes. The project and the Java class contained in it are compiled using Maven.

> **Note:**
>
> If Maven reports an error in resolving a dependency or artifact, add the repository into the POM file using the Repositories page and try again. Make sure that the repository that the URL points to is up and accessible.

How to Create a Maven POM for a Project

The `pom.xml` is the configuration file that contains the information to build a project. Use the procedure to create a Maven POM for a project from an existing project by adding elements and settings for the compiler.

You can create a Maven POM based on an existing project that you select in the Applications window. 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 for a project:

1. In the Applications window, 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 **Maven POM for Project.**
5. Complete the dialog and click **OK**.

Auditing Maven Applications

Empty tags or invalid values cause errors during a build run. An Audit handles the errors encountered when creating a Maven project for the `pom.xml`.

While editing `pom.xml`, empty tags or invalid values can cause errors when running the build. These errors need to be caught and reported as part of audit. Audit should handle any errors that are reported when creating a Maven project for the `pom.xml`.

Some examples for validations:

- Invalid/ empty values for artifact id, group id, version in dependency, plugins etc. after inheritance is applied.
- Duplicate entries in profile activation rules.

Configuring Test Settings

Specify how Maven finds test resources in an application by double-clicking the POM file and selecting Test Settings in Overview. Configure the application test source and resource paths using the JUnit Test project in JDeveloper, or configure the test paths from the project properties dialog for the test project.

You can specify how Maven finds test resources in your application. Double-click the POM file and select Test Settings in the Overview tab. You can manually configure application's test source and resource paths. In the Test Settings tab, you can specify a project for testing purposes or point JDeveloper to testing resources.

If there is a separate project for testing, JDeveloper can detect the paths to the various testing artifacts. If the testing resources are in a project with other source and artifacts, you must select those explicitly. More

There are two ways to configure test settings:

- If you already have a dedicated JDeveloper project (JUnit Test project) that contains the test sources and resources, then use that on the Test Settings tab.
- You can also configure the test paths from the project properties dialog for the test project. The test project sources and resources are automatically synced into the associated POM file.

Understanding Code Insight

Code insight speeds up the code writing in a `pom.xml` source file fetching required values from the local repository. Use the procedure to see code insight.

You can use code insight to speed up the process of writing code in your `pom.xml` source file.

To see code insight:

1. Open a `pom.xml` file in the Code Editor.
2. On the Plugins tab, add a plugin element.
3. Add the plugin's <executions> section.

Support for code insight is based on values in the Overview tab, which are fetched from the local repository. For example, for packaging type, code insight displays the list of values that is shown in the Overview editor for Packaging Type.

The following code insight lists of values are also supported:

- phases, inside plugin
- update policy, checksum policy and layout
- type and scope, inside dependency

Using the WebLogic Maven Plugin in JDeveloper

Use the WebLogic Maven Plugin to deploy, redeploy, undeploy, and update applications built using Maven to a WebLogic Server from within the Maven environment.

You can use the plug-in to deploy, redeploy, and update applications built using Maven to WebLogic Server from within the Maven environment. You can even undeploy an application. See *Deploying Applications to Oracle WebLogic Server*.

Using ojdeploy and ojmake

JDeveloper supports deployment in Ant scripts by adding a deploy target to the `build.xml` file. Modify the POM file to add the `ojdeploy` and `ojmake` deployment plugins to use commands with the Maven build tool, configure the plug-ins during POM creation, and add deployment profiles to the project.

JDeveloper supports deployment in Ant scripts. It does this by adding a deploy target to the `build.xml` file. See How to Deploy from the Command Line Using Ant .

You can use `ojmake`, as well as `ojdeploy`, in Maven POM files. JDeveloper modifies the POM file and adds `ojdeploy` and `ojmake` deployment plugins when necessary, as shown below:

```
<build>
  <plugins>
    <plugin>
        <groupId>com.oracle.adf.plugin</groupId>
        <artifactId>ojmake</artifactId>
        <version>12.1.3-0-0</version>
        <configuration>
          <ojmake>
            ${oracleHome}/jdeveloper/jdev/bin/ojmake
          </ojmake>
          <files>
            ${basedir}/Application2.jws
          </files>
          <usemaven>
            true
          </usemaven>
        </configuration>
        <executions>
          <execution>
            <phase>compile</phase>
            <goals>
              <goal>compile</goal>
            </goals>
          </execution>
        </executions>
      </plugin>
    <plugin>
```

```
      <groupId>oracle.jdeveloper.deploy.maven</groupId>
      <artifactId>maven-ojdeploy-plugin</artifactId>
      <version>1.0.0</version>
      <configuration>
        <ojdeploy>/scratch/jdoe/view_storage/pmed_jdev_l/oracle/jdeveloper/
jdev/bin/ojdeploy</ojdeploy>
        <workspace>/home/jdoe/jdeveloper/mywork/Application1/
Application1.jws                </workspace>
        <project>ViewController</project>
        <profile>Application1_ViewController_webapp</profile>
        <usemaven/>
      </configuration>
      <executions>
        <execution>
          <phase>install</phase>
          <goals>
            <goal>deploy</goal>
          </goals>
        </execution>
      </executions>
    </plugin>
  </plugins>
</build>
```

JDeveloper provides the **ojmake** and **ojdeploy** plug-ins. When you add these plug-ins, JDeveloper can do the following:

- Use the **ojmake** and **ojdeploy** commands with the Maven build tool
- Configure the plug-ins during POM creation
- Configure the plug-ins in the POM (if not already present) when you add deployment profile to the project

To add the plugins, double-click the project's POM file and click on the Plugins tab. Select the **ojmake** and **ojdeploy** plug-ins in the Build column and configure the parameters as needed in the Configuration section.

To see the usage, parameters, options, and examples for `ojmake`, at the command line for *`jdeveloper_install`*`/jdeveloper/jdev/bin` type `ojmake`.

Understanding Continuous Delivery and Continuous Integration

A Maven repository hosts artifacts such as compiled code and metadata about that code. Maven projects are built against the local repository, a cache of a remote repository that is stored on the local machine, and remote repository, a repository that contains all the Maven artifacts and plugins.

Continuous delivery is an extension of practices, from the build process through to the actual delivery of the software. Continuous delivery takes the next step to automate release management and deployment of software to end users, starting from where continuous integration finishes, automation of the building and testing of software. The purpose is to minimize the time between users expressing a requirement and a product being delivered to address that requirement. This avoids issues commonly seen in large software development projects.

Continuous integration is a software engineering practice which attempts to improve quality and reduce time to deliver software by applying small, frequent quality control efforts. It is characterized by these practices:

- Use of a version control system

- Developers commit to the main code line every day
- The product is built on every commit
- The build must be automated and fast
- Automated deployment to a production-like environment
- Automated testing is employed
- Results of all builds are published so everyone can see who broke the build
- Deliverables are easily available to developers, testers, and other stakeholders

10

Testing and Profiling Java Application Projects

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.
This chapter includes the following sections:

- About Testing and Profiling Java Application Projects
- Testing Java Application Projects with Unit Tests
- Starting a Profiling Session
- Taking and Accessing Snapshots of Profiling Data
- Taking a Heap Dump
- Setting a Profiling Point
- Profiling Telemetry
- Profiling Methods
- Profiling Objects
- Profiling Threads
- Profiling Locks
- Additional Functions when Running a Profiling Session

About Testing and Profiling Java Application Projects

The profiler gathers statistics on a running program that enable you to diagnose performance issues and correct code inefficiencies.

The IDE provides tools for creating and running unit tests and for profiling Java applications. Unit tests enable you to test the code in Java applications. Profiling is the process of examining an application to locate memory or performance-related issues.

When profiling a Java application, you can monitor the Java Virtual Machine (JVM) and obtain data about application performance, including method timing, object allocation and garbage collection. You can use this data to locate potential areas in your code that can be optimized to improve performance.

The following profiling capabilities are available:

- **Telemetry**—monitors CPU, memory usage, number of threads and loaded classes. See Profiling Telemetry.
- **Methods**—profiles methods execution times and invocation count, including call trees. See Profiling Methods.
- **Objects**—profiles size and count of allocated objects including allocation paths. See Profiling Objects.
- **Threads**-—profiles threads time and state. See Profiling Threads.
- **Locks**—profiles locks content data. See Profiling Locks.

Testing Java Application Projects with Unit Tests

JUnit is an open source regression testing framework for Java. Use JUnit to write and run tests that verify Java code.

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.

Creating a JUnit Test for a Java Project

A JUnit test application comprises one or more test cases, test fixtures, a test suite that invokes the cases, and a runner that invokes the suite.

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.

How to Create a JUnit Custom Test Fixture

A test fixture comprises a set of objects with known values that provides data for test cases. Use the procedure to create a JUnit custom test fixture class.

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.

> **Note:**
>
> `UnitTestFixture` is a public class. If you create an instance of it, there will be no errors in generated code.
>
> `AppModuleAMFixtur`e is a private class. If you create an instance of it, there will be errors in the generated code.

To create a JUnit custom test fixture class:

1. In the Applications window, select the project.
2. Choose **File** > **New** > **From 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. Modify the file as needed.

   In particular, add code that initializes test fixture objects to the `setUp()` method. Add code that releases any resources they acquire to the `tearDown()` method.

How to Create a JUnit JDBC Test Fixture

A test fixture provides data for the test cases, and a JDBC test fixture creates a database connection for the test cases. Use the procedure to create a JUnit JDBC test fixture class.

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 Applications window, select the project.
2. Choose **File** > **New** > **From Gallery**.
3. In the Categories tree, expand **General** and select **Unit Tests (JUnit)**.
4. In the Items list, double-click **Test Fixture**.
5. Complete the dialog to create the test fixture class.

   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.

Creating a JUnit Test Case

A test case class calls JUnit assertions to conduct tests by means of its methods. Use the procedure to create a JUnit test case class.

A test case class has one or more methods that perform tests by calling JUnit assertions. The following is 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.

```
public void testCountChars()
{
      int expected = 4;
      int actual = fixture1.countChars('a');
      assertEquals(expected, actual);
}

@Test
public void testCountChars()
{
      int expected = 4;
      int actual = fixture1.countChars('a');
      Assert.assertEquals(expected, actual);
}
```

In the test case above, `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 below:

```
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 Applications window, select the project or the particular class that you want to test.
2. Choose **File** > **New** > **From Gallery**.
3. In the Categories tree, expand **General** and select **Unit Tests**.
4. In the Items list, double-click **Test Case**.
5. In the Select the Class to Test page of the Create Test Case dialog, enter the class under test or click **Browse**.
6. In the Class Browser dialog, locate the class you want to test or enter the beginning letters in the **Match Class Name** field. The **Match Class** list will be filtered for easier identification.

   Select the class and click **OK** to close the dialog. Click **Next**.
7. Select the individual methods you want to test and click **Next**.
8. In the Setup Test Case Class page, enter the name of the test case, the package, and the class it extends and select the list of built-in functions JUnit will create stubs for. Click **Next**.
9. In the Select Test Fixtures page, select any test fixtures you want to add to the test case or click **Browse**.
10. Make sure that all the test fixtures you want to add to the test case are selected in the list and click **Finish**.

    The class created by the wizard will be opened for editing.

You can create a test case specifically for an EJB application. For more information, see How to Test EJB Unit with JUnit.

How to Add a Test to a JUnit Test Case

A unit test for a method can be added to a JUnit test case class. Use the procedure to add a test to a JUnit test case class.

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**.
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.

Creating a JUnit Test Suite

Insert a `main()` method and a call to a `TestRunner` class to see the test results displayed in the JUnit TestRunner log window. Use the procedure to create a JUnit test suite class.

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 below, 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.

```
public class AllTests {
    public static Test suite() {
    TestSuite suite;
    suite = new TestSuite("project1.AllTests");
    return suite;    }
```

In the JUnit 4 test suite shown below, the test case classes are written with `@Suite` and `@RunWith` annotations.

```
@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:

Before you create a JUnit test case, you must have created a project that is to be tested.

1. In the Applications window, select the project.
2. Choose **File** > **New** > **From 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 suite class. The class created by the wizard displays 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.

How to Create a Business Components Test Suite

Generate tests for every view object in the application module using the JUnit BC4J Test Suite wizard. Use the procedure to create a business components test suite.

The test fixture that is created is a singleton class to reduce the number of connections. If you want to connect or disconnect for each test case, customize the test case using the JUnit 4 annotations `@Before and @After`.

The JUnit BC4J Test Suite wizard will generate tests for each view object in the application module. If the application module does not have exported methods, the wizard will also generate a test for the application module itself. A generated view object class has the format `view_objectVOTest.java` and is placed into a package with the format `package.view.viewobjectVO`, where package is the application module package. A generated application module test has the format `application_moduleAMTest.java` and is placed into a package with the format `package.applicationModule`. A generated test fixture class has the format `applicationmoduleAMFixture.java` and is placed in the same package as the application module test.

The generated all test suite class has the format `AllapplicationmoduleTest.java` and is placed into the package with the same name as the application module package name.

A test case XML file is also generated for each application module or view object test. The XML file contains test methods defined in the application module or view object test cases. It does not include the test methods from the base classes (if any) because there may be too many duplicates.

To create a business components test suite:

1. In the main menu, choose **File** and then **New**.

   You will create a separate project for the business components tests.
2. In the New Gallery, expand **General**, select **Projects** and then **Java Projects**, and click **OK**.
3. In the Project Name page of the Create Java Project wizard, enter a name and the directory path for the test project, and click **Next**.
4. In the Project Java Settings page, enter the package name, the directory of the Java source code in your project, and output directory where output class files will be placed, and click **Finish**.
5. In the Applications window, double-click the application module you want to test.
6. In the overview editor, click the **Java** navigation tab.
7. In the Java page, click the **Edit** icon for the **Java Class** section.
8. In the Select Java Options dialog, select **Generate Application Module Class** and click **OK**.
9. In the Java page of the overview editor, click the **Edit** icon for the **Class Interface** section.
10. In the Edit Client Interface dialog, shuttle the methods you want to test to the **Selected** pane, and click **OK**.
11. In the Applications window, right-click the test project you have created and choose **New**.
12. In the New Gallery, expand **General**, select **Unit Tests** and then **Business Components Test Suite**, and click **OK**.
13. In the Configure Tests page of the JUnit BC4J Test Suite wizard, select values for the following and click **Next**:
    - **Business Component Project**: Select the project that has the application module you want to test.
    - **Application Module**: Select the application module you want to test.
    - **Configuration**: Choose a local or shared application module.
    - **Test Base Class-Application Module Extends**: You can specify different base cases. The generated test case classes will extend from that base class where all public

abstract methods in the base class will have simple and default implementation method bodies.

- **Test Base Class-View Object Extends**: You can specify which class the view object extends. The generated test case classes will extend from that base class where all public abstract methods in the base class will have simple and default implementation method bodies.

14. In the Summary page, verify the selections and click **Finish**.

How to Create a Business Components Test Fixture

Create a Business Components test fixture independently, or create it in conjunction with a business components test suite. Use the procedure to create a business components test fixture.

When you create a business components test suite, a business components test fixture is created with it. You can also create Business Components test fixtures independently.

A generated test fixture class has the format `applicationmoduleAMFixture.java` and put into a package with the format `package.applicationModule`, where package is the application module package.

To create a business components test fixture:

1. In the main menu, choose **File** > **New** > **From Gallery.**

   You will create a separate project for the business components tests.

2. In the New Gallery, expand **General**, select **Projects** > **Java Project**, and click **OK**.
3. In the Project Name page of the Create Java Project dialog, enter a name and the directory path for the test project, and click **Next**.
4. In the Project Java Settings page, enter the package name and the source and output directories, and click **Finish**.
5. In the Applications window, double-click the application module you want to test.
6. In the overview editor, click the **Java** navigation tab and then click the **Edit** icon for the Java Class section.
7. In the Select Java Options dialog, select **Generate Application Module Class**, and click **OK**.
8. In the Java page of the overview editor, click the **Edit** icon for the **Class Interface** section.
9. In the Edit Client Interface dialog, shuttle the methods you want to test to the **Selected** pane, and click **OK**.
10. In the Applications window, right-click the test project you have created and choose **New**.
11. In the New Gallery, expand **General**, select **Unit Tests** and then **Business Components Test Fixture**, and click **OK**.
12. In the Configure Tests page of the JUnit BC4J Test Fixture wizard, select values for the following and click **Next**:
    - **Business Component Project**: Select the project that has the application module you want to test.
    - **Application Module**: Select the application module you want to test.
    - **Configuration**: Choose a local or shared application module.
13. In the Summary page, verify the test fixture class and click **Finish**.

How to Update a Test Suite with all Test Cases in the Project

Update a test suite with all the test cases in a project. Use the procedure to update a test suite.

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 context menu, choose **Source** > **Refresh Test Suite**.
2. Ensure that all items in the list of test cases are checked.
3. To update the test suite, click **OK.**

How to Run JUnit Test Suites

You can run a successfully compiled test suite. Use the procedure to run a JUnit test suite.

When your test suite has been successfully compiled you can run it.

To run a JUnit test suite:

1. In the Applications window, select the test suite class.
2. Right click it, and choose **Run**.

   The test executes and the test runner displays the results.

Starting a Profiling Session

Learn different methods to start the profiler.

JDeveloper provides the following pathways for starting the profiler:

- Starting and Profiling JDeveloper Applications Simultaneously
- Attaching the Profiler to a Running JDeveloper Applications
- Profiling External Applications

Starting and Profiling JDeveloper Applications Simultaneously

Use the procedure to start a JDeveloper application and the profiling of that application.

You may simultaneously start a JDeveloper application and the profiling of that application by following these steps:

1. Choose **Run** > **Profile Project**
2. On the main window click the **Configure Session** button and select a profiler mode by clicking on it. You can change the profiler mode at any point by clicking the **Profile** drop down arrow.
3. On the main window, click the **Profile** button.

   The application and the profile session are simultaneously started.

How to Calibrate the Profiler

The calibration process has to be completed for each JDK that is used for profiling before the IDE can be used to profile an application. Use the procedure to calibrate the IDE on your local system.

You must calibrate the IDE before you can use the IDE to profile an application. You must run the calibration process for each JDK that you use for profiling. You do this because instrumenting the bytecode of the application imposes some overhead, and the time spent in code instrumentation needs to be "factored out" to achieve more accurate results.

You only have to calibrate the IDE once for each JDK that you use. However, you should run the calibration process again when anything changes on your local or remote configuration that could affect system performance. The following could affect system performance:

- Any hardware upgrade
- Any significant change or upgrade of the operating system
- An upgrade of the Java platform used for profiling

To calibrate the IDE on your local system:

1. Close any other programs that are running.

   The IDE runs the calibration if other applications are running, but running any CPU-intensive programs when performing the calibration might affect the accuracy of profiling results.

2. Go to **Tools** > **Preferences** > **Profiler** > **Manage Calibration Data** and click the **Manage** button.

   The **Manage Calibration Data** window appears.

3. Select the Java Platform to be used for profiling. Click **Calibrate**.

   You can click **Java Platforms** to open the Manage Libraries window to add a new Java platform. The **Manage Calibration Data** dialog box displays the date that the most recent calibration was performed.

When you click Calibrate, the IDE collects calibration data on the selected Java platform. When the calibration process is complete you can start using the IDE to profile your applications.

Do not share calibration data between various computers or systems.

Attaching the Profiler to a Running JDeveloper Applications

Use the procedure to profile a JDeveloper project that is already running.

You may profile a JDeveloper project that is already running by following these steps:

1. Choose **Run** > **Attach to Project**
2. On the main window click the **Configure Session** button and select a profiler mode by clicking on it. You can change the profiler mode at any point by clicking the **Profile** drop down arrow.

   If the target application is running outside JDeveloper, select the **Setup attach to project** target and make the appropriate selections in the **Attach Settings** window.

3. On the main window, click the **Attach** button.

A profile session tracking the running application is started.

Profiling External Applications

Use the procedure to profile an application that is not started from within the JDeveloper IDE.

You may profile an application that is not started from within the JDeveloper IDE by following these steps:

1. Choose **Run** > **Attach to External Process**

   The **Profiling External Process** tab opens in the main window.

2. Click the **Configure Session** button and select **Setup attach to process**

   The **Attach Settings** window appears.

3. Select the type of application to be profiled from the **Profile** drop-down menu and select the process to be profiled. The **Attach Settings** window explains and provides detailed instructions for each option. Click **OK**.

4. On the main window click the **Configure Session** button and select a profiler mode by clicking on it. You can change the profiler mode at any point by clicking the **Profile** drop down arrow.

5. Click the **Attach** button.

   A profile session tracking the selected application is started.

Taking and Accessing Snapshots of Profiling Data

Take snapshots during a profiling session, or at the end of the profiling session to get profiling data at a point in time, and view it using the Snapshot window.

A snapshot captures profiling data at a specific point in time and allows you to access them via the Snapshot window. See Accessing Snapshots

A snapshot differs from live profiling results in the following ways:

- Snapshots can be examined when no profiling session is running.
- Snapshots can be easily compared.

There are two options for taking snapshots:

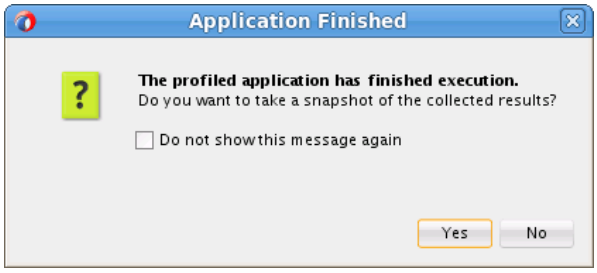
- While the profiling session is in progress. See Taking Snapshots During a Profiling Session
- At the end of the profiling session. See Taking Snapshots at the End of a Profiling Session

Taking Snapshots at the End of a Profiling Session

At the end of a profiling session, the system displays the Application Finished dialog, which you can use to take a snapshot of the results that are collected.

When closing a profiled application, or if it finishes on its own, while the profiling session is in progress, the profiler asks you whether to take a snapshot of the results collected so far by displaying the **Application Finished** dialog.

Figure 10-1 Application Finished Dialog



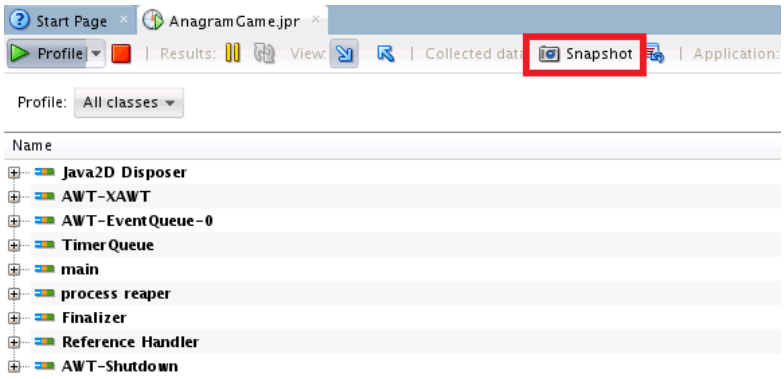
Click **Yes** to save the snapshot.

Taking Snapshots During a Profiling Session

Use the Snapshot option to take a snapshot of profiling data at any time during the profiling session.

You may take a snapshot of the profiling data at any time during the profiling session by clicking the **Snapshot** icon shown in the figure below.

Figure 10-2 Snapshot Icon



To control how the snapshots functionality behaves during a session, go to **Tools** > **Preferences** > **Profiler** and click the **When taking snapshots drop-down menu** to see the following options:

Open New Snapshot—it opens the snapshot right after clicking the Snapshot icon

Save New Snapshot—it saves a new snapshot every time you click the Snapshot Icon

Open and Save New Snapshot—it saves and opens a snapshot right after clicking the Snapshot icon.

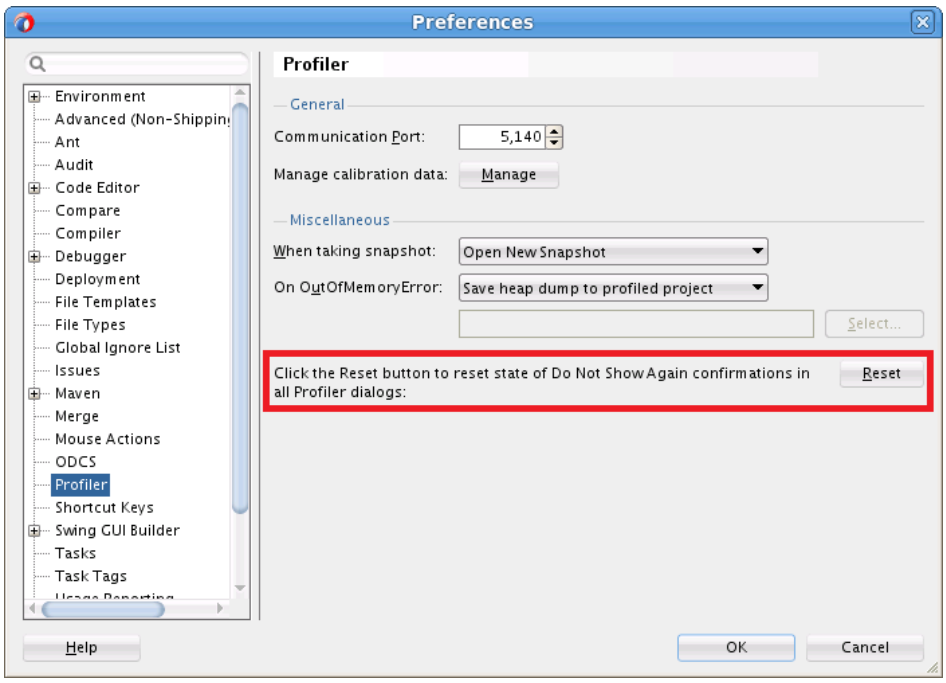
You may take multiple snapshots during a profiling session and you will also be prompted to save a "final" snapshot at the end of the session.

Starting and Stopping the Application Finished Dialog

Use the Profiler option available on the Preference menu for Tools to reset the Application Finished dialog that has been closed.

When the **Application Finished** dialog appears at the end of the profiling session, if you select the **Do not show this message again** checkbox the dialog would not display again. If at a later time you want to reactivate the display of this dialog, go to **Tools** > **Preference** > **Profiler** and click **Reset** button as shown in the figure below.

Figure 10-3 Reset Button in the Profiler Preferences Dialog

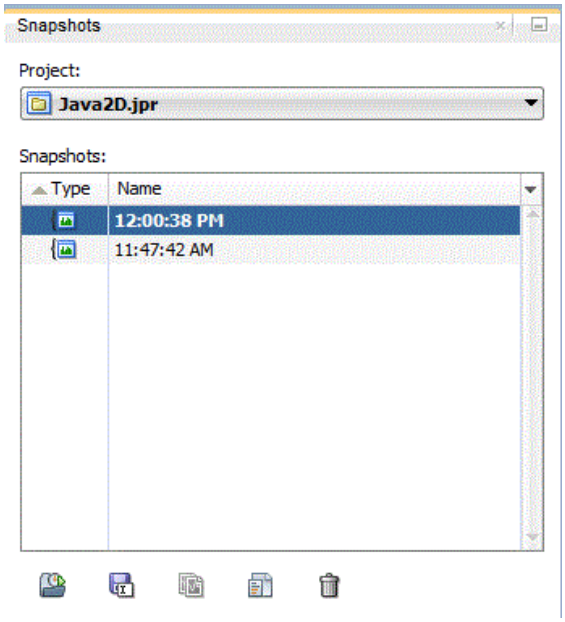


Accessing Snapshots

The Snapshots window accessed by means of the Snapshots option on the Profiling menu for Window displays the snapshots that were captured for a profiling session.

You may access you profiling session snapshots by going to **Windows** > **Profiling** > **Snapshots**. The **Snapshots** window appears as shown in the figure below.

Figure 10-4 Snapshot Windows



At the bottom of the **Snapshots** window there are icons that allow you to export, open, rename, and delete selected snapshots.

Taking a Heap Dump

Take a heap dump during a profiling session, and save the heap to a project or local file system so that the dump can be loaded and the objets in the heap dump browsed outside of a running profiling session. Use the procedure to take a heap dump using a profiling point.

You can take a heap dump when a profiling session is in progress. When you take a heap dump you are prompted to save the heap to your project or local file system. After you save a heap dump you can load the heap dump at any time and browse the objects on the heap, locate references to individual objects and compare heap dumps to view the differences between the snapshots. You do not need to have a running profiling session to load and browse the heap dump.

The application must be running on JDK 1.5.0_12 or higher to take a heap dump.

To take a heap dump using a profiling point:

1. Open the source file containing the code where you want to place the profiling point.
2. Right-click in the line of code where you want to place the profiling point and select **Add Profiling Point**.
3. In the **Profiling Point Type** list, select one of the following snapshot options and click **Next**:
   - Take Snapshot
   - Timed Take Snapshot
   - Triggered Take Snapshot

4. In the Customize Properties page of the wizard, select Heap Dump as the type of snapshot and modify any additional settings. The Heap Dump option is available under **Settings > Take**.

When you use a profiling point to take a heap dump, you specify the point in your source code where you want to place the profiling point. For example, you may want to take a heap dump when a thread enters a specific method.

To take a heap dump on OutOfMemory error:

1. From the Main menu, select **Tools** > **Preferences** > **Profiler**.
2. In the On OutOfMemoryError list, select an option from the drop-down list to specify what the IDE does when an OutOfMemoryError is encountered.

   The default behavior is to save the heap dump to the profiled project.

Viewing UI Elements with Heap Walker

Use the heap dump viewer, Heap Walker, to identify the exact location of the application UI. It displays the logical values of objects such as String value, or File path, and also provides a visual snapshot of UI attributes and elements.

The heap dump viewer (Heap Walker) displays logical values of objects such as String value, File path, URL address, etc. In addition, it also provides a visual snapshot of UI attributes and elements such as Color, Font, Button, etc.

For most classes and instances represented in the heap dump, the textual and numerical properties are adequate for describing and examining data structures and for discovering bugs such as memory leaks, inefficient memory usage and others. However, for many types of objects, the in-memory representation is not suited for quickly determining what the object is.

The visual representation feature is ideal for examining UI elements, where displaying object properties is not precise enough to aid users in identifying the exact location of the application UI. For example, by just reading position, size and references to nested elements of an UI container the user may not realize that an object may represent an Open File dialog created by the application.

Access the image representation of a heap dump element by browsing through the instance of a given class. Figure 10-5 shows Heap Walker panels including a visual preview of the application window at the point when the heap has been dumped.

Figure 10-5 Heap Walker - Image Preview

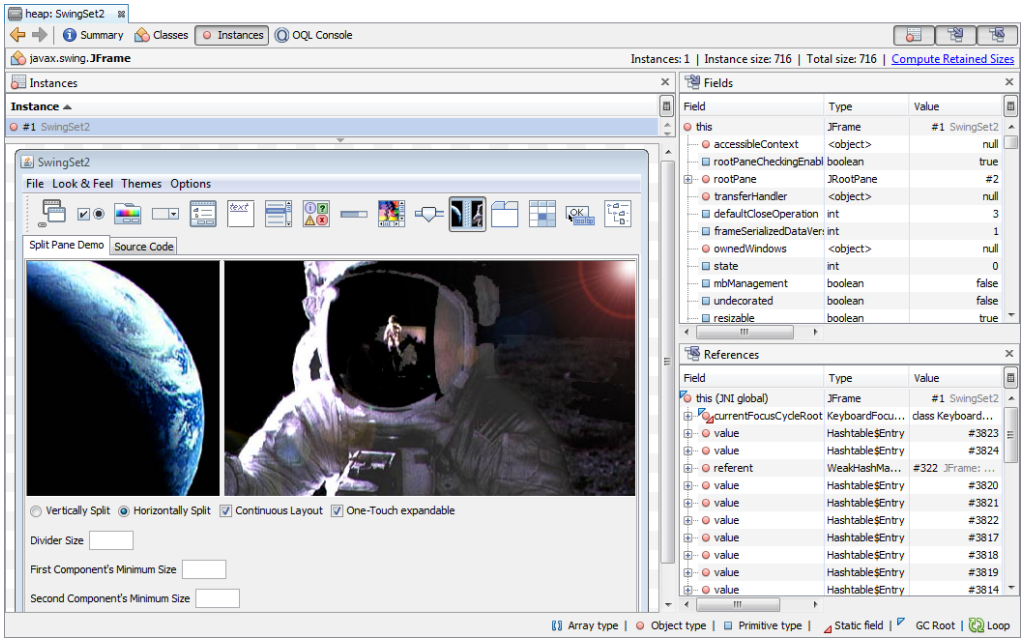


Image Preview Use Cases

Use the image preview feature in Heap Walker to identify a selected UI element, determine the application state at the time of dumping the heap, search for UI snippets in memory, discover duplications, and also to analyze the UI offline.

The Heap Walker image preview is useful in the following use cases:

- **Identifying selected UI element.** Browse instances of the desired type when you need to find a particular UI element like Button, Label, etc
- **Determining application state at the time of dumping the heap**. Bugs reported by users often do not contain all the necessary information or miss important details. By displaying the application UI users can immediately see that for example a text document was being loaded when an out of memory exception was thrown.
- **Searching for UI snippets unintentionally kept in memory.** Parts of the UI are sometimes not being released from memory, which can cause serious problems given that tables, trees or editors often reference very large data models. By browsing for example Panel elements users can easily discover these snippets, realize how much memory is being wasted and identify the problem preventing the UI from being released.
- **Discovering duplicities**. By browsing Images, for example, users can immediately see multiple instances of the same image being allocated in memory, which is a waste of resources.
- **Offline UI analysis.** Heap Walker is able to recreate the UI structure from a heap dump. This way an UI developer can analyze the UI building blocks without access to the actual application, which could be running on a different and incompatible system

It is important to note the following exceptions to the Image Preview function:

- No support for viewing tree data.
- Foreground, background or font attributes may not show on certain implementations.

- Custom controls cannot be displayed.
- UI text for certain elements may not fully display in the Instance view.

How to Analyze a Heap Dump Using Object Query Language (OQL)

Object Query Language, based on JavaScript expression language, queries a Java heap to filter content. Open the OQL editor when you load a Java heap in the Heap window by clicking the OQL Console tab of the window.

OQL is a SQL-like query language to query a Java heap that enables you to filter/select information wanted from the Java heap. While pre-defined queries such as "show all instances of class X" are already supported by the tool, OQL adds more flexibility. OQL is based on JavaScript expression language.

When you load a Java heap in the Heap window, you can click the OQL Console tab of the window to open the OQL editor. The OQL Console contains an OQL editor, a saved OQL queries window and a window that displays the query results. You can use any of the sample OQL queries or create a query to filter and select heap data to locate the information that you want from the Java heap. After you choose or write a query, you can run the query against the Java heap and view the results.

An OQL query is of the following form:

```
select <JavaScript expression to select>
[ from [instanceof] <class name> <identifier>
[ where <JavaScript boolean expression to filter> ] ]
```

where class name is fully qualified Java class name (example: `java.net.URL`) or array class `name. char[]` (or [C) is char array name, `java.io.File` (or `[Ljava.io.File;`) is name of `java.io.File[]` and so on. Note that fully qualified class name does not always uniquely identify a Java class at runtime. There may be more than one Java class with the same name but loaded by different loaders. So, class name is permitted to be id string of the class object. If instanceof keyword is used, subtype objects are selected. If this keyword is not specified, only the instances of exact class specified are selected. Both from and where clauses are optional.

In select and (optional) where clauses, the expression used in JavaScript expression. Java heap objects are wrapped as convenient script objects so that fields may be accessed in natural syntax. For example, Java fields can be accessed with `obj.field_name` syntax and array elements can be accessed with `array[index]` syntax. Each Java object selected is bound to a JavaScript variable of the identifier name specified in from clause.

OQL Examples

Study the examples to use OQL to query heaps.

Select all Strings of length 100 or more:

```
select s from java.lang.String s where s.count >= 100
```

Select all int arrays of length 256 or more:

```
select a from int[] a where a.length >= 256
```

Show content of Strings that match a regular expression:

```
select {instance: s, content: s.toString()} from java.lang.String s
    where /java/(s.toString())
```

Show path value of all File objects:

```
select file.path.toString() from java.io.File file
```

Show names of all `ClassLoader` classes:

```
select classof(cl).name
    from instanceof java.lang.ClassLoader cl
```

Show instances of the Class identified by given id string:

```
select o from instanceof 0xd404b198 o
```

0xd404b198 is id of a Class (in a session). This is found by looking at the id shown in that class's page.

OQL built-in objects and functions

The list provides the built-in objects that are supported by heaps.

Heap object

The heap built-in object supports the following methods:

- `heap.forEachClass` - calls a callback function for each Java Class

  ```
  heap.forEachClass(callback);
  ```

- `heap.forEachObject` - calls a callback function for each Java object

  ```
  heap.forEachObject(callback, clazz, includeSubtypes);
  ```

  `clazz` is the class whose instances are selected. If not specified, defaults to `java.lang.Object. includeSubtypes` is a boolean flag that specifies whether to include subtype instances or not. Default value of this flag is true.

- `heap.findClass` - finds Java Class of given name

  ```
  heap.findClass(className);
  ```

  where `className` is name of the class to find. The resulting Class object has following properties:

  - name - name of the class.
  - superclass - Class object for super class (or null if `java.lang.Object`).
  - statics - name, value pairs for static fields of the Class.
  - fields - array of field objects. field object has name, signature properties.
  - loader - ClassLoader object that loaded this class.

  Class objects have the following methods:

  - `isSubclassOf` - tests whether given class is direct or indirect subclass of this class or not.
  - `isSuperclassOf` - tests whether given Class is direct or indirect superclass of this class or not.
  - `subclasses` - returns array of direct and indirect subclasses.
  - `superclasses` - returns array of direct and indirect superclasses.

- `heap.findObject` - finds object from given object id

  ```
  heap.findObject(stringIdOfObject);
  ```

- `heap.classes` - returns an enumeration of all Java classes
- `heap.objects` - returns an enumeration of Java objects

  ```
  heap.objects(clazz, [includeSubtypes], [filter])
  ```

  `clazz` is the class whose instances are selected. If not specified, defaults to `java.lang.Object. includeSubtypes` is a boolean flag that specifies whether to include subtype instances or not. Default value of this flag is true. This method accepts an optional filter expression to filter the result set of objects.
- `heap.finalizables` - returns an enumeration of Java objects that are pending to be finalized.
- `heap.livepaths` - return an enumeration of paths by which a given object is alive. This method accepts optional second parameter that is a boolean flag. This flag tells whether to include paths with weak reference(s) or not. By default, paths with weak reference(s) are not included.

  ```
  select heap.livepaths(s) from java.lang.String s
  ```

  Each element of this array itself is another array. The later array is contains an objects that are in the 'reference chain' of the path.
- `heap.roots` - returns an Enumeration of Roots of the heap.

  Each Root object has the following properties:

  – id - String id of the object that is referred by this root
  – type - descriptive type of Root (JNI Global, JNI Local, Java Static, etc.)
  – description - String description of the Root
  – referrer - Thread Object or Class object that is responsible for this root or null

Examples

- Access static field 'props' of class `java.lang.System`

  ```
  select heap.findClass("java.lang.System").statics.props
  select heap.findClass("java.lang.System").props
  ```
- Get number of fields of `java.lang.String` class

  ```
  select heap.findClass("java.lang.String").fields.length
  ```
- Find the object whose object id is given

  ```
  select heap.findObject("0xf3800b58")
  ```
- Select all classes that have name pattern java.net.*

  ```
  select filter(heap.classes(), "/java.net./(it.name)")
  ```

Functions on individual objects

- `allocTrace` function

  Returns allocation site trace of a given Java object if available. `allocTrace` returns array of frame objects. Each frame object has the following properties:

  – className - name of the Java class whose method is running in the frame.
  – methodName - name of the Java method running in the frame.
  – methodSignature - signature of the Java method running in the frame.

  - sourceFileName - name of source file of the Java class running in the frame.
  - lineNumber - source line number within the method.

- `classof` function

  Returns class object of a given Java object. The resulting object supports the following properties:

  - name - name of the class
  - superclass - class object for super class (or null if `java.lang.Object`)
  - statics - name, value pairs for static fields of the class
  - fields - array of field objects. Field objects have name, signature properties
  - loader - ClassLoader object that loaded this class.

  Class objects have the following methods:

  - `isSubclassOf` - tests whether given class is direct or indirect subclass of this class or not
  - `isSuperclassOf` - tests whether a given class is direct or indirect superclass of this class or not
  - subclasses - returns array of direct and indirect subclasses
  - superclasses - returns array of direct and indirect superclasses

  Examples

  - Show class name of each Reference type object

    ```
    select classof(o).name from instanceof java.lang.ref.Reference o
    ```

  - Show all subclasses of `java.io.InputStream`

    ```
    select heap.findClass("java.io.InputStream").subclasses()
    ```

  - Show all superclasses of `java.io.BufferedInputStream`

    ```
    show all superclasses of java.io.BufferedInputStream
    ```

- `forEachReferrer` function

  Calls a callback function for each referrer of a given Java object.

- `identical` function

  Returns whether two given Java objects are identical or not, for example:

  ```
  select identical(heap.findClass("Foo").statics.bar,
  heap.findClass("AnotherClass").statics.bar)
  ```

- `objectid` function

  Returns String id of a given Java object. This id can be passed to `heap.findObject` and may also be used to compare objects for identity. For example:

  ```
  select objectid(o) from java.lang.Object o
  ```

- `reachables` function

  Returns an array of Java objects that are transitively referred from the given Java object. Optionally accepts a second parameter that is comma separated field names to be excluded from reachability computation. Fields are written in `class_name.field_name` pattern.

  Examples

- Print all reachable objects from each Properties instance.

  ```
  select reachables(p) from java.util.Properties p
  ```

- Print all reachables from each `java.net.URL` but omit the objects reachable via the fields specified.

  ```
  select reachables(u, 'java.net.URL.handler') from java.net.URL u
  ```

- `referrers` function

  Returns an enumeration of Java objects that hold reference to a given Java object. This method accepts optional second parameter that is a boolean flag. This flag tells whether to include weak reference(s) or not. By default, weak reference(s) are not included.

  Examples

  - Print number of referrers for each `java.lang.Object` instance

    ```
    select count(referrers(o)) from java.lang.Object o
    ```

  - Print referrers for each `java.io.File` object

    ```
    select referrers(f) from java.io.File f
    ```

  - Print URL objects only if referred by 2 or more

    ```
    select u from java.net.URL u where count(referrers(u)) > 2
    ```

- `referees` function

  Returns an array of Java objects to which the given Java object directly refers to. This method accepts optional second parameter that is a boolean flag. This flag tells whether to include weak reference(s) or not. By default, weak reference(s) are not included. For example, to print all static reference fields of `java.io.File` class:

  ```
  select referees(heap.findClass("java.io.File"))
  ```

- `refers` function

  Returns whether first Java object refers to second Java object or not.

- `root` function

  If the given object is a member of root set of objects, this function returns a descriptive Root object describing why it is so. If given object is not a root, then this function returns null.

- `sizeof` function

  Returns size of the given Java object in bytes, for example:

  ```
  select sizeof(o) from int[] o
  ```

- `retainedsize` function

  Returns size of the retained set of the given Java object in bytes. **Note**: Using this function for the first time on a heap dump may take significant amount of time.

  The following is an example usage of the `retainedsize` function:

  ```
  select rsizeof(o) from instanceof java.lang.HashMap o
  ```

- `toHtml` function

  Returns HTML string for the given Java object. Note that this is called automatically for objects selected by select expression. But, it may be useful to print more complex output. For example, to print a hyperlink in bold font:

  ```
  select "<b>" + toHtml(o) + "</b>" from java.lang.Object o
  ```

Selecting Multiple Values

Use JavaScript object literals or arrays to select multiple values.

Multiple values can be selected using JavaScript object literals or arrays.

For example, show the name and thread for each thread object

```
select { name: t.name? t.name.toString() : "null", thread: t }
from instanceof java.lang.Thread t
```

array/iterator/enumeration manipulation functions

These functions accept an array/iterator/enumeration and an expression string [or a callback function] as input. These functions iterate the array/iterator/enumeration and apply the expression (or function) on each element. **Note**: JavaScript objects are associative arrays. So, these functions may also be used with arbitrary JavaScript objects.

- `concat` function

  Returns whether the given array/enumeration contains an element the given boolean expression specified in code. The code evaluated can refer to the following built-in variables.

  - it - currently visited element
  - index - index of the current element
  - array - array/enumeration that is being iterated

  For example, to select all Properties objects that are referred by some static field some class:

  ```
  select p from java.util.Properties p
  where contains(referrers(p), "classof(it).name == 'java.lang.Class'")
  ```

- `count` function

  Returns the count of elements of the input array/enumeration that satisfy the given boolean expression. The boolean expression code can refer to the following built-in variables.

  - it - currently visited element
  - index - index of the current element
  - array - array/enumeration that is being iterated

  For example, print the number of classes that have a specific name pattern:

  ```
  select count(heap.classes(), "/java.io./(it.name)")
  ```

- `filter` function

  Returns an array/enumeration that contains elements of the input array/enumeration that satisfy the given boolean expression. The boolean expression code can refer to the following built-in variables.

  - it - currently visited element
  - index - index of the current element
  - array - array/enumeration that is being iterated
  - result -> result array/enumeration

  Examples

- Show all classes that have `java.io.*` name pattern

  ```
  select filter(heap.classes(), "/java.io./(it.name)")
  ```

- Show all referrers of URL object where the referrer is not from java.net package

  ```
  select filter(referrers(u), "! /java.net./(classof(it).name)")
  from java.net.URL u
  ```

- `length` function

  Returns number of elements of an array/enumeration.

- `map` function

  Transforms the given array/enumeration by evaluating given code on each element. The code evaluated can refer to the following built-in variables.

  - it - currently visited element
  - index - index of the current element
  - array - array/enumeration that is being iterated
  - result -> result array/enumeration

  `Map` function returns an array/enumeration of values created by repeatedly calling code on each element of input array/enumeration.

  For example, show all static fields of `java.io.File` with name and value:

  ```
  select map(heap.findClass("java.io.File").statics, "index + '=' + toHtml(it)")
  ```

- `max` function

  Returns the maximum element of the given array/enumeration. Optionally accepts code expression to compare elements of the array. By default numerical comparison is used. The comparison expression can use the following built-in variables:

  - lhs - left side element for comparison
  - rhs - right side element for comparison

  Examples

  - Find the maximum length of any string instance

    ```
    select max(map(heap.objects('java.lang.String', false), 'it.count'))
    ```

  - Find string instance that has the maximum length

    ```
    select max(heap.objects('java.lang.String'), 'lhs.count > rhs.count')
    ```

- `min` function

  Returns the minimum element of the given array/enumeration. Optionally accepts code expression to compare elements of the array. By default numerical comparison is used. The comparison expression can use the following built-in variables:

  - lhs - left side element for comparison
  - rhs - right side element for comparison

  Examples

  - Find the minimum size of any vector instance

    ```
    select min(map(heap.objects('java.util.Vector', false), 'it.elementData.length'))
    ```

  - Find vector instance that has the maximum length

```
select min(heap.objects('java.util.Vector'), 'lhs.elementData.length <
rhs.elementData.length')
```

- `sort` function

  Sorts a given array/enumeration. Optionally accepts code expression to compare elements of the array. By default numerical comparison is used. The comparison expression can use the following built-in variables:

  – lhs - left side element for comparison

  – rhs - right side element for comparison

  Examples

  – Print all char[] objects in the order of size.

  ```
  select sort(heap.objects('char[]'), 'sizeof(lhs) - sizeof(rhs)')
  ```

  – Print all char[] objects in the order of size but print size as well.

  ```
  select map(sort(heap.objects('char[]'), 'sizeof(lhs) - sizeof(rhs)'),  '{ size:
  sizeof(it), obj: it }')
  ```

- `top` function

  Returns top N elements of the given array/enumeration. Optionally accepts code expression to compare elements of the array and the number of top elements. By default the first 10 elements in the order of appearance is returned. The comparison expression can use the following built-in variables:

  – lhs - left side element for comparison

  – rhs - right side element for comparison

  Examples

  – Print 5 longest strings

  ```
  select top(heap.objects('java.lang.String'), 'rhs.count - lhs.count', 5)
  ```

  – Print 5 longest strings but print size as well.

  ```
  select map(top(heap.objects('java.lang.String'),  'rhs.count - lhs.count', 5),
  '{ length: it.count, obj: it }')
  ```

- `sum` function

  Returns the sum of all the elements of the given input array or enumeration. Optionally, accepts an expression as second param. This is used to map the input elements before summing those.

  For example, return the sum of sizes of the reachable objects from each Properties object:

  ```
  select sum(map(reachables(p), 'sizeof(it)'))
  from java.util.Properties p

  // or omit the map as in ...
  select sum(reachables(p), 'sizeof(it)')
  from java.util.Properties p
  ```

- `toArray` function

  Returns an array that contains elements of the input array/enumeration.

- `unique` function

  Returns an array/enumeration containing unique elements of the given input array/ enumeration.

The following example selects a unique char[] instances referenced from strings. Note that more than one string instance can share the same char[] for the content.

```
// number of unique char[] instances referenced from any String
select count(unique(map(heap.objects('java.lang.String'), 'it.value')))

// total number of Strings
select count(heap.objects('java.lang.String'))
```

Other Examples

The examples show how to print a histogram of each class loader and the number of classes loaded by it, and the parent-child chain for each class loader instance.

The following example prints a histogram of each class loader and number of classes loaded by it.

`java.lang.ClassLoader` has a private field called classes of type `java.util.Vector` and Vector has a private field named `elementCount` that is number of elements in the vector. The query selects multiple values (loader, count) using JavaScript object literal and map function. It sorts the result by count (i.e., number of classes loaded) using sort function with comparison expression.

```
select map(sort(map(heap.objects('java.lang.ClassLoader'),
'{ loader: it, count: it.classes.elementCount }'), 'lhs.count < rhs.count'),
'toHtml(it) + "<br>"')
```

The following example shows the parent-child chain for each class loader instance.

```
select map(heap.objects('java.lang.ClassLoader'),
      function (it) {
         var res = '';
         while (it != null) {
            res += toHtml(it) + "->";
            it = it.parent;
         }
         res += "null";
         return res + "<br>";
      })
```

Note that the parent field of `java.lang.ClassLoader` class is used and the example walks until the parent is null using the callback function to map call.

The following example prints the value of all System properties. Note that this query (and many other queries) may not be stable - because private fields of the Java platform classes may be modified or removed without any notification (implementation detail). But using such queries on user classes may be safe, given that you have control over the classes.

```
select map(filter(heap.findClass('java.lang.System').props.table, 'it != null && it.key !
= null && it.value != null'),
            function (it) {
                var res = it.key.toString() + ' = ' + it.value.toString();
                return res;
            });
```

- `java.lang.System` has static field by name 'props' of type `java.util.Properties`.
- `java.util.Properties` has field by 'table' of type `java.util.Hashtable$Entry` (this field is inherited from `java.util.Hashtable`). This is the hashtable buckets array.

- `java.util.Hashtable$Entry` has key, value and next fields. Each entry points the next entry (or null) in the same hashtable bucket.
- `java.lang.String` class has a value field of type char[].

Setting a Profiling Point

Set a profiling point in source code to invoke specific profiling actions. Use the procedure to set a profiling point, and view the active profiling points.

A profiling point is a marker in your source code which can invoke specific profiling actions. You set a profiling point in your code by using the popup menu in the Source Editor or by using the toolbar in the Profiling Points window.

You can set the following types of profiling points:

- Reset Results
- Stopwatch
- Take Snapshot
- Timed Take Snapshot
- Triggered Take Snapshot

Note: Icons for the Timed Take Snapshot and Triggered Take Snapshot do not display in code editors. They only display in the Profiling Points window.

You can use a profiling point to reset profiling results, take a snapshot or record the timestamp or execution time of a code fragment.

Once you set a profiling point it becomes part of the project until you delete it.

To set a profiling point:

1. Locate the class where you want to add the profiling point and open the class in the Source Editor.
2. In the Source Editor, right-click in the gutter on the line where you want to add the profiling point.
3. Select **Add Profiling Point** to open the New Profiling Point wizard.
4. Select a profiling point type and the project.
5. Click **Next**.
6. Customize the properties of the profiling point, if necessary.
7. Click **Finish**.

An icon representing the profiling point type appears in the Source Editor where you inserted the profiling point.

To enable or disable a profiling point, in the Source Editor, right-click in the left margin of the line containing the profiling point and choose **<Profiling point name>** > **Enable** or **Disable**.

To view active profiling points:

1. Open the application.
2. From the main menu, select **Run** > **Profile**.. The last profile mode session is displayed. and a **Profiling Points** section appears.

   Alternatively you may select **Window** > **Profiling** > **Profiling Points**.

3. Use the Project or Profiling Point columns to view the defined profiling points.

Profiling Telemetry

The telemetry mode provides the metrics of CPU and GC, Memory, Surviving Generations, and Threads and Classes.

The telemetry mode provides the following metrics:

CPU and GC—displays the CPU and GC percentage of use at a given time

Memory—displays in MB the heap size and used heap at a given time

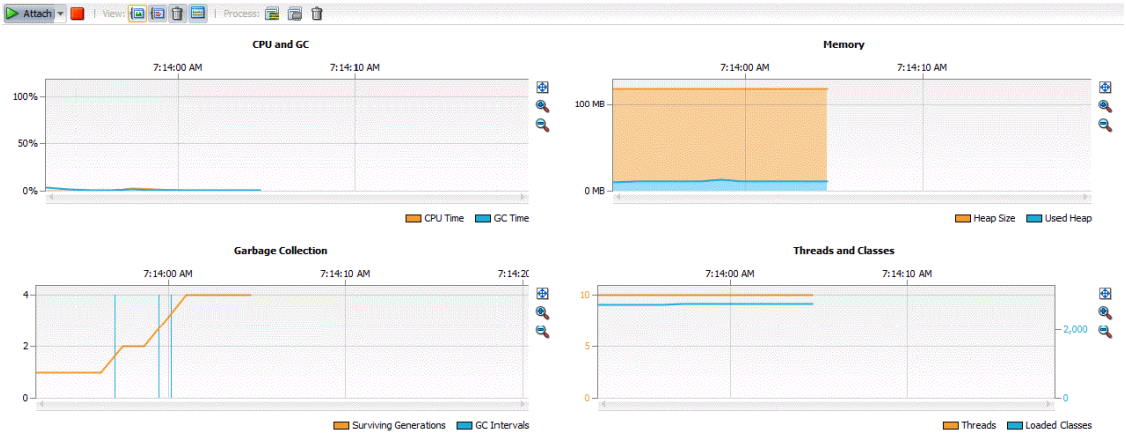
Surviving Generations —displays the number of surviving generations at a given time. It also displays indicates the GC intervals

Threads and Classes—displays number of loaded classes and threads at a given time

To start a profiling telemetry session, see "About Starting the Profiler"

Figure 10-6 shows a snapshot of a telemetry session.

Figure 10-6 Telemetry Session



Profiling Methods

Use the methods mode to view metrics for methods and classes. The methods report provides icons to view the data by Forward Calls, Hot Spots, Reverse Calls, Show Delta Value, and Select Threads.

The methods mode provides metrics for methods and classes. The methods report allows you to view the data by **Forward Calls**, **Hot Spots**, and **Reverse Calls** by clicking on the appropriate icons. You may also choose to **Show Delta Values** and **Select Threads**.

- **Show Delta Values**-this action switches from absolute values to incremental values. The values displayed prior to switching the view are remembered but the new view displays changes starting at the moment the new selection was made. Clicking this icon again resets the results back to absolute values.
- **Select Threads**-this action shows threads available in live results or a saved snapshot and allows you to select specific threads for displaying results. This feature is especially useful when tracking EDT slowness in desktop applications or analyzing worker threads in server

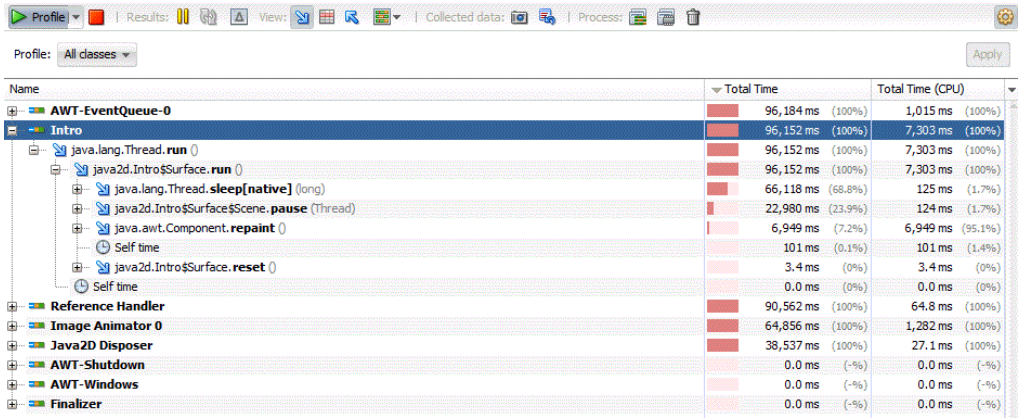
applications. The **Merge selected threads** option is enabled if some of the threads are selected and it is disabled for the **Show all threads** option. This feature merges results from the selected threads to a single tree.

Additionally, you may select the columns to be displayed; the options are **Total Time**, **Total Time (CPU)**, **Selected**, and **Hits/Invocations** (depending on the session configuration).

To start a profiling methods session, see "About Starting the Profiler"

Figure 10-7 shows a snapshot of a methods session.

Figure 10-7 Methods Session



Profiling Specific Methods

Use the procedure to narrow the scope of the methods session to profile specific methods or classes.

You may narrow down the scope of the methods session to profile specific methods or classes by following these steps:

1. Click on the icon.

   A new settings bar appears.
2. On the **Profile** dropdown make the appropriate selection.
3. Click the **Plus** icon to add a class or method.

   The **Select Class or Select Method** window appears.
4. To select a class in the **Select Class** window, choose the **Project**, **Package**, and **Class** and click **OK**.

   To select a method in the **Select Method** window, choose the **Project**, **Package**, **Class** and **Method** and click **OK**

   Alternatively, you may right click on a method or a class to select it. This feature is available in the profile results and snapshot windows.

Profiling Objects

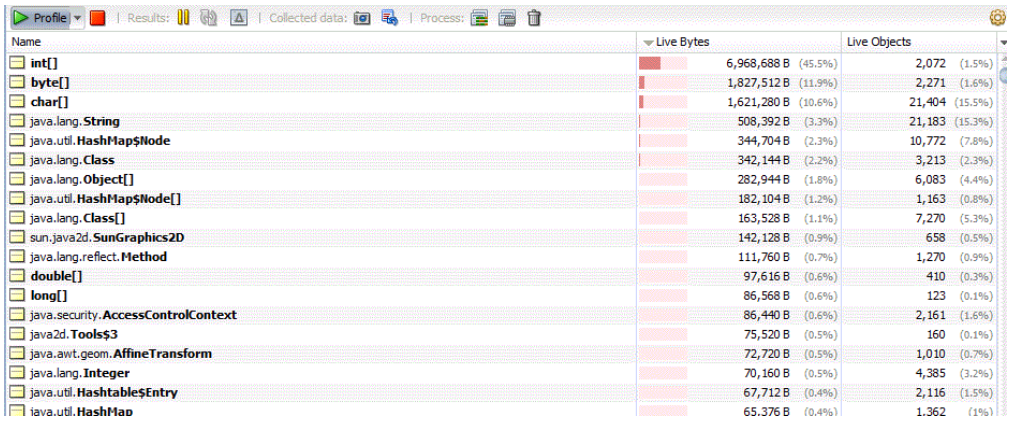
Use the objects mode to view the list of classes allocated to a project.

The objects mode provides a list of classes allocated to a project including live instances and bytes allocation. By clicking the icon on the top-right corner of the page you access a drop-down menu that allows you select the classes to be profiled. The **All Classes** mode shows all classes and object that are live on the Virtual Machine heap. The **Project Classes** filter allows to view only the classes defined in the project.

To start a profiling objects session, see "About Starting the Profiler"

Figure 10-8 shows a snapshot of an Objects session

Figure 10-8 Objects Session



Profiling Specific Objects

Use the Profile Class option that is available on the context menu of a class to profile a specific class.

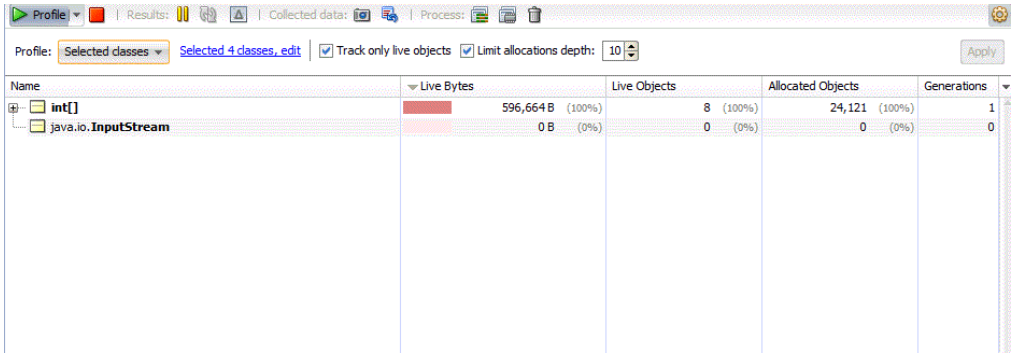
You may elect to profile specific classes by right clicking on a class and selecting **Profile Class.** The **Track only live objects** and **Limit allocations depth** checkboxes appear.

Track only live objects — when selected, it tracks only live objects. If not selected, it tracks all objects allocated by the application.

Limit allocations depth — limits the stack depth allocations to the number specified.

After selecting a class, click the **Apply** button on the right while the profiling session is in progress to submit your changes. This action clears the view to display only the classes that you selected as shown in Figure 10-9

Figure 10-9 Objects Session - Selected Classes View



Profiling Threads

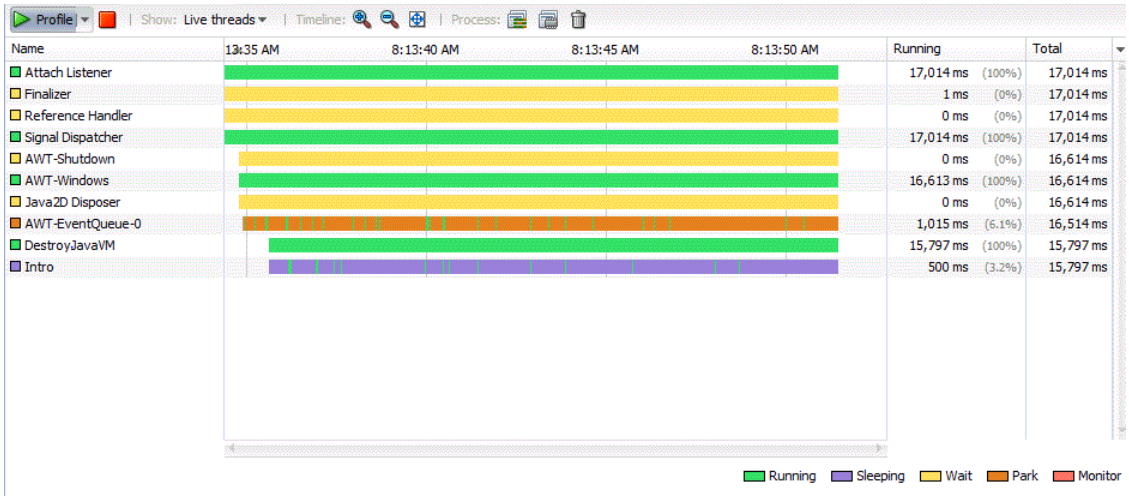
View the detailed information about application thread activity using the threads mode. Customize threads using the options that are available on the Live Threads list displayed in a profiling threads session.

The threads mode allows you to view detailed information about application thread activity.

To start a profiling threads session, see "About Starting the Profiler"

Figure 10-10 shows a snapshot of a threads session

Figure 10-10 Threads Session



Additionally, you may customize the threads you monitor by accessing the Live **Threads** drop-down list and choosing from the available options: **All Threads**, **Live Threads** (Default), **Finished Threads**, and **Selected Threads**.

Profiling Locks

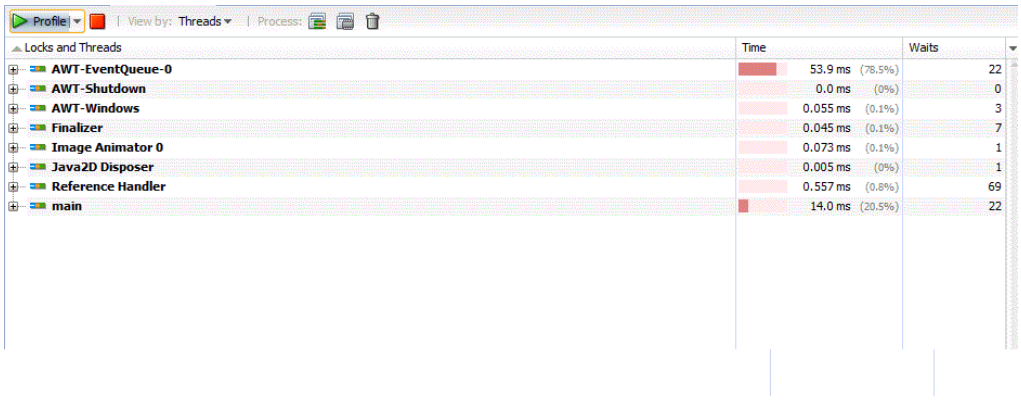
View details about locked threads, and the threads that are monitoring and holding locks using the locks mode. In a profiling locks session, the Threads list shows the locked threads, and Monitors shows the threads that are locking other threads.

The locks mode allows you to view details about locked threads and the threads that are monitoring and holding locks.

To start a profiling locks session, see "About Starting the Profiler"

Figure 10-11 shows a snapshot of a locks session

Figure 10-11 Locks Session



In the session window you can choose **Threads** or **Monitors** in the **Threads** drop-down list. Choose **Threads** to view locked threads. Expand the nodes to view the owners of the locks. Choose **Monitors** to view the threads that are locking other threads.

Additional Functions when Running a Profiling Session

The toolbar of the profiler window during a profiler session shows actions that pertain to the actual profiler mode such as Thread dump, Heap dump, GC, Snapshot, and Reset collected results.

While the profiler session is in progress, additional actions related to the actual profiler mode are available in the toolbar of the profiler window. The following actions are always available:

Thread dump—creates a textual dump of all active threads and monitors of the profiled application. It shows what methods have been executed at the point of capturing the dump, thread by thread. This information is useful to view what the application is currently doing. The thread dump also contains information about locks, threads holding the locks, and threads waiting to acquire a lock. This data is essential when debugging deadlocks. To capture a Thread Dump, click the T**hread Dump** icon during the profiling session. To learn more about taking snapshots, see Taking and Accessing Snapshots of Profiling Data

Heap dump—saves an image of the current heap content of the profiled process in .hprof format and optionally opens it in heap browser. See Capturing Heap Dump Data

GC—requests the JVM of the profiled process to invoke garbage collection. The JVM behavior for garbage collection is not defined in the JVM specification. It should do the garbage collection at some point, but there is no guarantee it will do it immediately or at all.

Additionally, when profiling Methods or Objects, the following actions are available:

Snapshot—creates a snapshot of all currently collected profiling data related to methods or objects. The snapshot opens in a separate window and can be saved to the project or to an external file. See Taking and Accessing Snapshots of Profiling Data

Reset collected results—clears all currently collected profiling data related to methods or objects.

The other actions displayed in the toolbar of the profiler window are specific to the actual profiling mode. If multiple profiling modes are active in a profiling session, the toolbar displays actions available for the currently displayed modes.

11

Auditing and Monitoring Java Projects

This chapter describes the auditing and monitoring capabilities of Oracle JDeveloper. This chapter includes the following sections:

- About Auditing and Monitoring Java Projects
- Auditing Java Projects
- Monitoring HTTP Using the HTTP Analyzer

About Auditing and Monitoring Java Projects

Use the auditing and monitoring tools to analyze the performance of applications. An auditing feature analyzes Java code in JDeveloper for conformance to programming standards, and the HTTP Analyzer analyzes the HTTP traffic between projects to facilitate debugging an application.

Use the auditing and monitoring 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.

Auditing is concerned with programming standards, rather than syntactic correctness. You can audit code even when it is not compilable or executable.

You can use the HTTP Analyzer to facilitate debugging your application in terms of the HTTP traffic sent and received between your projects' web service clients and services and between your Java applications and web resources.

Auditing Java Projects

JDeveloper supports audit features such as qualitative tests for the presence or absence of some feature called rules, and the quantitative measurement of size or complexity through metrics. Use the listed methods to audit Java code.

Auditing is the static analysis of code for adherence to rules and metrics that define programming standards. A software code audit is a comprehensive analysis of source code in a programming project with the intent of discovering bugs, security breaches or violations of programming conventions.

- 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.

JDeveloper'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. See How to Run Audit to Generate an Audit Report.
- Use Code Assist to audit while editing. Code Assist enables background audits while you edit. Audit violations are highlighted as you edit. You can apply automated corrections.
- Audit from the command line to produce an audit report. See Auditing Java Code from the Command Line.
- Display the Issues window. The Issues 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.

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.

Understanding Audit Rules

Audit rules are static, qualitative, analyses of code. Use the tabulated properties to enable and configure the properties for individual rules in an auditing profile.

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.

The rules contain the properties shown in Table 11-1.

Table 11-1 Rule Properties

Property	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.
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.

Understanding Audit Metrics

JDeveloper measures the metrics that are static, quantitative analyses of code. Configure metrics by setting a threshold, and enable and configure individual metrics within auditing profiles.

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 metrics shown in Table 11-2.

Table 11-2 Audit Metrics

Metric	Description
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.

Using the Auditing Tools

Auditing is the static evaluation of code against programming standards. Use auditing tools to view audit reports in the Log window, and to rectify violations and over-threshold measurements.

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.

Using the Audit Window Report Panel

An audit report presents rule violations and metrics measurements as a tree organized into constructs: methods, classes, packages, files, projects, or workspaces. An audit profile with rules includes a Severity column, and an audit profile with metrics includes a measurements column.

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.

Using the Audit Window Toolbar

Use the Log window toolbar for operations such as refresh, cancel, and export.

From the Log window toolbar you can perform the operations shown in Table 11-3.

Table 11-3 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.
	Group 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.
	Show Anomalies Only	Toggle the display of measurements that are within acceptable limits. The threshold is a settable property of metrics.
	Show Suppressed Issues	Toggle to show the suppression scheme issues.
	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 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.

Using the Audit Window Context Menu

In the Audit window, right-click one or more selected constructs or rule violations to open the context menu, and to use operations such as create, cancel, refresh, and export.

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 11-4 on the selected constructs or rule violations.

Table 11-4 Audit Window Context Menu Items

Name	Description
Create <*construct*>	Choose to apply the specified fix (constant, static field, instance field, variable, or method).
About <*construct*> Rule	Choose to display an explanation of the rule that applies to this rule violation.
Hide <*Rule*> Issues	Choose to remove all violations of the selected rule from the report.
Show Hidden Issues	Choose to restore all previously hidden issues.
Show Anomalies Only	Click to toggle the display of measurements that are within acceptable limits.
Show Error Issues	Click to toggle the display of the number of errors in the selected file, or to list the errors in the file
Show Warning Issues	Click to toggle the display of the number of warnings in the selected file, or to list the warnings in the file.
Show Incomplete Issues	Click to toggle the display of the number of incomplete issues in the selected file, or to list the incomplete issues in the file.
Show Advisory Issues	Click to toggle to show the display of the number of advisory issues in the selected file, or to list the advisory issues in the file.
Show Suppressed Issues(X)	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.

How to Audit Java Code in JDeveloper

Use the JDeveloper auditing tools to find and fix defects in code. Use the procedure to audit Java Code.

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. See Working with Audit Profile

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** <*project*>.
   - You can also audit Java code from the command line by invoking ojaudit.exe, which is included in your JDeveloper installation.
3. Inspect the completed Audit Report for rule violation.

   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.
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.

You can save the finished audit report as an XML file or as a formatted HTML or text file. 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.

Auditing Java Code from the Command Line

Invoking the `ojaudit.exe` from the command line audits a workspace, a project, or a source file. Tabulated are the parameters you can use during an audit.

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 11-5 contains the parameters you can use during the audit.

Table 11-5 Command Line Parameters

Parameter	Description
`file`	Specifies the workspace (`.jws`), project (`.jpr`), or source (`.java`) file to be audited.
`-classpath` *`path`*	Sets the class path for files to audit, if a project is not being audited.
`-disable` *`name`*	Disables the specified rule or metric in profile. To supply multiple values, repeat this option. This option requires the use of `-profile`.
`-enable` *`name`*	Enables the specified rule or metric in the profile. To supply multiple values, repeat this option.
`-encoding` *`code`*	Sets 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).
`-fail` *`severity`*	Sets the issue severity that the Auditor will regard as failure.
`-fix`	Applies default fixes to the code. This option modifies source files.
`-help`	Prints help for the command help and exits.

Table 11-5 (Cont.) Command Line Parameters

Parameter	Description
`-listall`	Lists all audited files in the audit report including those that have no issues.
`-maxfilesize` *size*	Specifies the maximum file size, in Mb, to audit.
`-metric` *name*	Enables the specified metric.
`-nometric` *name*	Disables the specified metric. This option requires the use of `-profile` .
`-norule` *name*	Disables the specified audit rule. This option requires the use of `-profile`.
`-notitle`	Sets an empty audit report title.
`-output` *file*	Specifies the pathname of the output file. If omitted, output is written to standard output.
`-profile` *name*	(required) Specifies 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.
`-profileoutput` *file*	Sets the output file for the merged profile.
`-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.
`-quiet`	Suppresses the copyright message.
-role name	Sets the active JDeveloper customization role.
-rule name	Enables the specified rule.
-seal	Seals the specified profiles. This option requires the use of `-profile` with an explicit path is also used.
-rulehelp	Prints the available rules and exits.
`-sourcepath` *path*	Set source path for files to audit, if a project is not being audited
`-style` *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.
`-verbose`	Causes all execution messages to be displayed.
`-version`	Prints the command's version and exits.
`-workingset` *name*	Sets the working set for files to audit. This option requires the use of `-workspace`.
`-workingsethelp`	Prints the available working sets for the workspace and exits.
`-workspace` *file*	Sets the workspace context for files to audit.

Table 11-5 (Cont.) Command Line Parameters

Parameter	Description
`-xmlinput` *file*	Changes the format of an existing XML report. This option requires the use of `-style`.
`-xmloutput` *file*	Sets the output file as a plain XML report. This option requires the use of `-style`.
`@file`	Includes options and parameters from the audit file.

Note the following considerations:

- Unless the *file* specified is a workspace or project, you must specify `-project`, `-sourcepath`, or `-classpath`.
- If a project depends on other projects in the workspace, you must specify `-workspace`.
- The options `-profile` and `-style` accept a name or a URL. Case and whitespace in the name are ignored.
- The options `-enable` and `-disable` accept an ID or a label. Case and whitespace in the name are ignored.
- The options `-rule` and `-metric` are synonymous of -enable. The options `-norule` and `-nometric` are synonymous of -`disable`.

Working with Audit Profile

An audit profile specifies the rules, code assists, and metrics that will be used to analyze Java code. Use a predefined profile, or modify a profile to create a custom profile.

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. You can activate and deactivate rules, code assists, and metrics for an audit profile from the Audit Profiles preferences page. While several profiles are predefined, you can create others by modifying an existing profile. 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, as shown in Table 11-6.

Table 11-6 Audit Profile

Profile	Description
Code Assist Rules	Used by the Source Editor, Issues window, Application Overview, and File List.
Compile Rules	Used at the end of a compile when Audit While Compiling is selected in the Audit page of the Preferences dialog
Audit Rules	Used by the Source Editor, Issues window, Application Overview, and File List. This is the initial default for the Audit command. However, this is not permanent because the Audit dialog remembers whatever profile was last selected.
Javadoc Rules	Used by the Source Editor.

Table 11-6 (Cont.) Audit Profile

Profile	Description
ADF Best Practice Rules	Used for ADF applications.

JDeveloper provides predefined profiles, each with different combinations of the available rules, code assists, and metrics:

- ADF Best Practice Rules
- All Metrics
- All Rules
- Audit Rules
- Code Assist Rules
- Compile Rules
- Javadoc Rules

Create an Audit Profiler

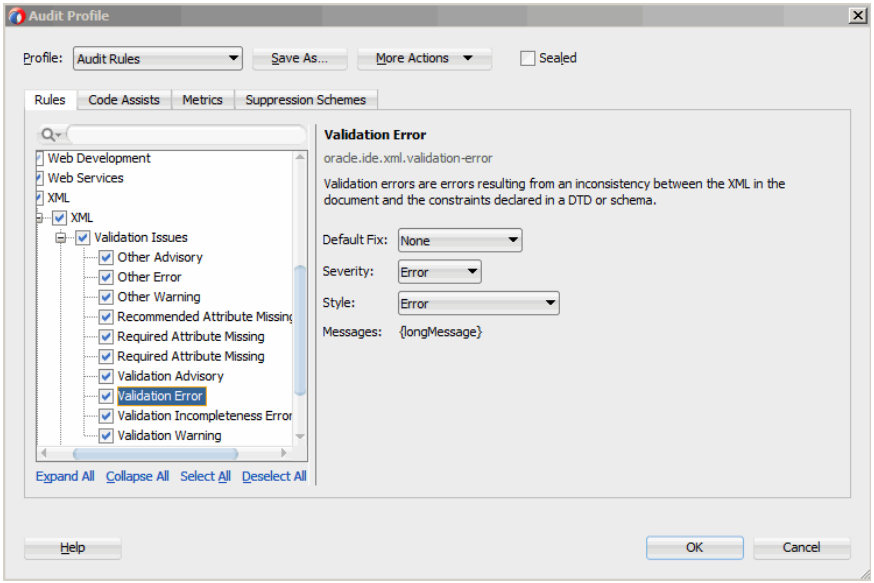
Use the procedure to create an audit profile.

To create an audit profile:

1. From the main menu, choose **Tools** > **Preferences**.
2. Choose **Audit** and click **Manage Profiles** on the Audit page.
3. From the Profile drop-down list, choose a profile to copy.

   As shown in the figure, the selected profile's property names and current values display in the right panel. A description of the selected item displays in the Explanation box. Its properties and settings display in the right pane.

Figure 11-1 Audit Profile Page



4. Select the rules, assists, and metrics to enable in the new profile.
5. Click **Save As**.
6. 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.

7. Click **OK**.

Sealing a Profile

Saving the audit profile with the Sealed option selected in the Audit Profile page seals the profile so that only the selected rules are enabled when the audit profile is next loaded. Sealing the audit profile preserves the project environment, and prevents new audit issues from being introduced.

By saving the audit profile with the Sealed option selected in the Audit Profile page, you seal the current audit profile, which means that only the rules selected for that profile are enabled the next time that specific audit profile is loaded. Any new rules that are present since the profile was last saved are disabled (regardless of their default state). New rules can be introduced in a number of ways, such as when new extensions are installed when JDeveloper is updated. In the course of developing a project, sealing the audit profile preserves the project environment, preventing new audit issues from being introduced.

Disabling Suppression Schemes

A suppression scheme suppresses issues discovered through a project audit, and disabling suppression schemes reduces auditing time, and reduces Log output. Use the procedure to disable a suppression scheme.

A **suppression scheme** describes a scheme for suppressing issues (that is, audit violations) discovered through a project audit. When auditing a project, you can view suppression issues in the Audit Log window. You can enable (the default state) or disable suppression schemes from being audited through the Audit Profile dialog. Disabling suppression schemes can reduce auditing time and reduce output in the Log window.

To disable a suppression scheme:

1. From the main menu, choose **Tools** > **Preferences**.
2. Choose **Audit** and click **Manage Profiles** on the Audit page.
3. Click the **Suppression Scheme** tab. Uncheck the top-level node to disable all the suppression schemes of a specific category or expand the nodes to disable a specific scheme.

   Check a box to enable a suppression scheme.
4. Click **OK** in the dialog.

How to Delete an Audit Profile

Although a predefined profile cannot be deleted, an existing audit profile, and any custom profile that was created with the Save As command can be deleted. Use the procedure to delete an existing audit profile.

You can delete an existing audit profile, but not a predefined profile. That any custom profile you created with the Save As command, you can delete.

To delete an existing audit profile:

1. From the main menu, choose **Tools** >**Preferences**.

   The Preferences dialog opens.
2. Choose **Audit** and click **Manage Profiles** on the Audit page.
3. From the **Profile** drop-down list, choose a custom profile to be deleted.
4. From the **More Actions** drop-down list, choose **Delete**.

   The profile is removed from the **Profile** box. Note that the predefined profiles provided by the IDE are grayed out and cannot be deleted.
5. Click **OK**.

How to Import or Export an Audit Profile

Importing or exporting audit profiles as XML files allows profile sharing, or maintaining a checked in profile used by `ojaudit` and a nightly build. Use the procedure to import or export an audit profile.

You can import or export audit profiles. This enables 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. Choose **Audit** and click **Manage Profiles** on the Audit page.
3. From the **More Actions** drop-down list, choose **Import** or **Export**, and select the profile you want to import or export.

How to Run Audit to Generate an Audit Report

Generate an audit report when you audit your Java programs so that rule violations and over-threshold measurements can be rectified. Use the procedure to generate an audit report.

When you audit your Java programs, you can generate an audit report. An audit report is a list of rule violations and over-threshold measurements. In the audit report, you can investigate these problems, and manually or automatically correct them.

To generate an audit report:

1. In the Applications window, 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** > **Audit <*target*>**, for example, **Build** > **Audit helloWorld.java**.
3. In the Audit <*target*> dialog, choose a profile to use in one of the two following ways:
   - From the **Profile** drop-down list 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.

How to Audit Unserializable Fields

Implementing the `java.io.Serializable` interface serializes an object making it possible to compress it into bytes so that it can be inflated later. Use the procedure to set audit rules.

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:

1. From the main menu, choose **Tools** > **Preferences** > **Audit** > **Manage Profiles**.
2. Click the Rules tab and expand the nodes, **Java SE** > **Java** > **Serialization**.
3. Check Non-Serializable Field in Serializable Class.

   A description of the rule is shown in right panel. You can set the default fix for the violation, the severity level of the violation, and the style of warning.
4. Click OK.

How to Audit Serializable Fields That Do Not Have serialVersionUID

The identifier, `serialVersionUID` enables versioning but even without this identifier, you can run an audit to flag the classes that implement `java.io.Serializable`. Use the procedure to set audit rules.

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:

1. From the main menu, choose **Tools** > **Preferences** > **Audit** > **Manage Profiles**.
2. Click the Rules tab and expand the nodes, **Java SE** > **Java** > **Serialization**.
3. Check Missing Serialization UID.

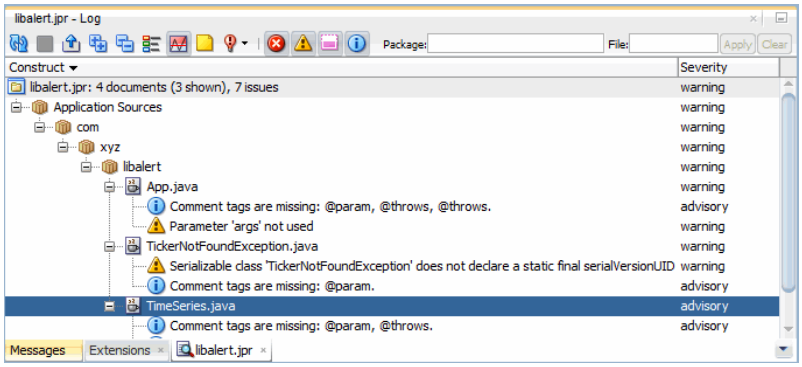
   A description of the rule is shown in right panel. You can set the default fix for the violation, the severity level of the violation, and the style of warning.
4. Click **OK**.

Viewing an Audit Report

JDeveloper generates a report of all audit rule violations and over-threshold measurements so that they can be corrected. Use the procedures to refresh an audit report, and to inspect an audit rule violation.

JDeveloper generates a report of all audit rule violations. Use the audit report to investigate and correct rule violations and over-threshold measurements. As shown in the figure, audit reports are displayed as tabbed panes of the Log window. In this window, you can choose a fix for a rule violation from a drop-down menu. For an individual rule violation, choose among the fixes defined for that violation's type.

Figure 11-2 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**.

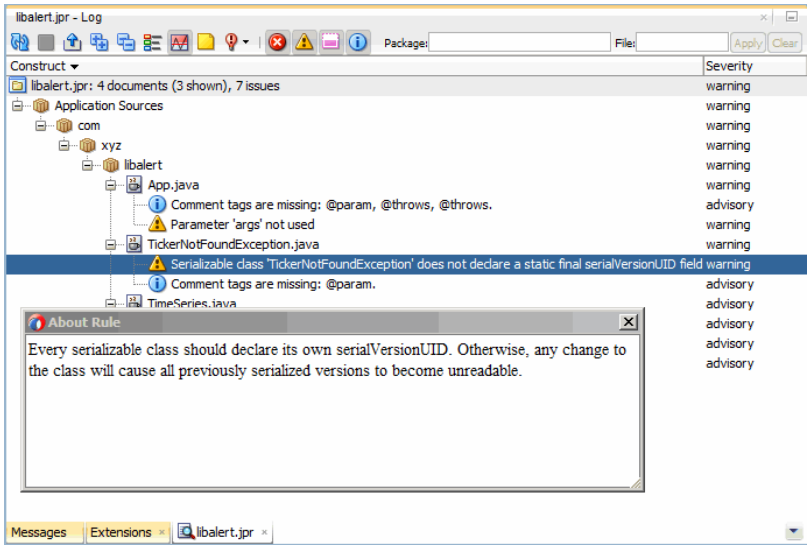
The Export Audit Results dialog clears, and a new audit begins. If you want to stop the audit, click in the Log's toolbar.

To inspect an audit rule violation:

1. In the audit report, select the construct you want 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** *violation* **Rule** to learn more about the rule that has been violated. A dialog describing the rules displays, as shown in the figure below.

Figure 11-3 About Rule Dialog



You can rearrange the audit report columns into left or right positions.

To organize audit report columns:

- Drag the column headers left or right to your preferred position.

How to Organize Audit Report Rows

An Audit report is organized as a tree with each row corresponding to a construct or a violation that includes measured values or a theoretical violation. Use the procedure 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.
2. Select the constructs you want to see.
3. Click **OK**.
4. Click a column header to sort rows by that column.

   To reverse the sort order, click again.

Using Filters with Reports

Patterns in filters determine which classes pass through the filters, and specifying filters by package names, class names, or both prunes the set of Java classes whose violations are displayed. Tabulated are the filters that you can specify.

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 11-7 contains the filters you can specify

Table 11-7 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.

How to Filter Audit Report Rows

Use the procedure to filter audit report rows. Enter the sequence of patterns to be applied to all but the last element of FQCNs in the Package field and enter the sequence of patterns for the last element in Field.

To filter audit report rows:

1. In the **Package** field 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** field, 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 redisplays to show only the selected rows.
4. Click **Clear** to delete text from the Package and File boxes.

How to Save an Audit Report

Save audit reports as XML files or as formatted HTML or text files. Use the step 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 directory, *jdev_install*/`jdev`/*system*/`audit`/`stylesheets` (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 display. Choose a title, format, and destination for the report, and click **OK**.

How to Fix an Audit Rule Violation

Edit source code to fix audit rule violations manually, or select an automated fix; use defined fixes for individual rules; and use Apply Default Fixes for a group construct. Use the procedures to fix an audit rule violation manually, and to apply an automated fix to 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. 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.

To fix an audit rule violation manually:

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 **Apply <*Rule*> Fix** menu items.

How to Fix a Construct's Audit Rule Violations

Constructs that have a Default Fix property with a value other than None will have automated fixes applied to each rule violation. Use the procedure to fix audit rule violations of a construct.

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**.

How to Hide Audit Rule Violations

Hiding all violations of a particular rule in the audit report makes the report more readable. Use the procedures to hide audit rule violations, and to restore hidden audit rule violations.

You can suppress the display of all the rule violations of a given type in the audit report. This may make the report easier to read, since it hides all violations of a particular rule. 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* **Issues**.

   All of the violations of the audit 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 Issues**.

   All of the previously hidden rule violations are restored to the audit report.

How to Hide Audit Report Measurements

Set the focus of metrics reports on over-threshold measurements by hiding other measurements. Use the Show Anomalies Only option in the Log window to show only over-threshold 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 Show Anomalies Only 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.

Monitoring HTTP Using the HTTP Analyzer

The HTTP Analyzer acts as a proxy between the code in JDeveloper and the HTTP resource, and thus helps in debugging an application.

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.

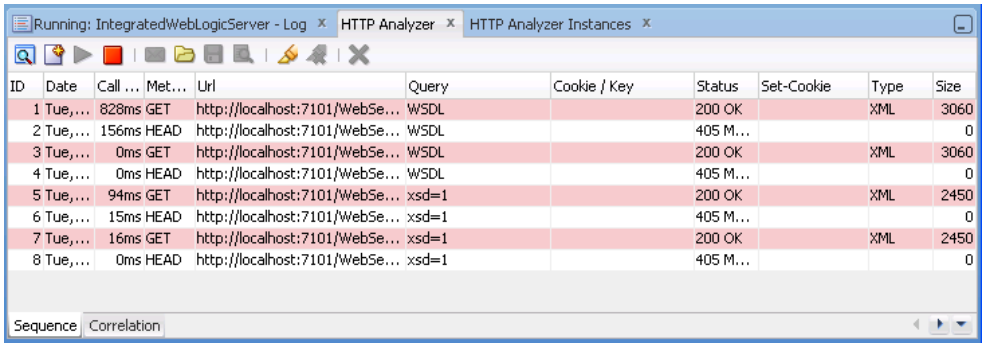
When you run the HTTP Analyzer, there are a number of windows that provide information for you.

How to Use the Log Window

The HTTP Analyzer Log window opens when you open the HTTP Analyzer from the Tools menu. Use the tools on the HTTP Analyzer Log Window Toolbar to reorder or group messages in the Log window.

When you open the HTTP Analyzer from the Tools menu, the HTTP Analyzer Log window opens, illustrated in the figure. By default its position is at the bottom center of JDeveloper, alongside the other log windows.

Figure 11-4 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 11-8 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.
	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 11-8 (Cont.) HTTP Analyzer Log Window Toolbar Icons

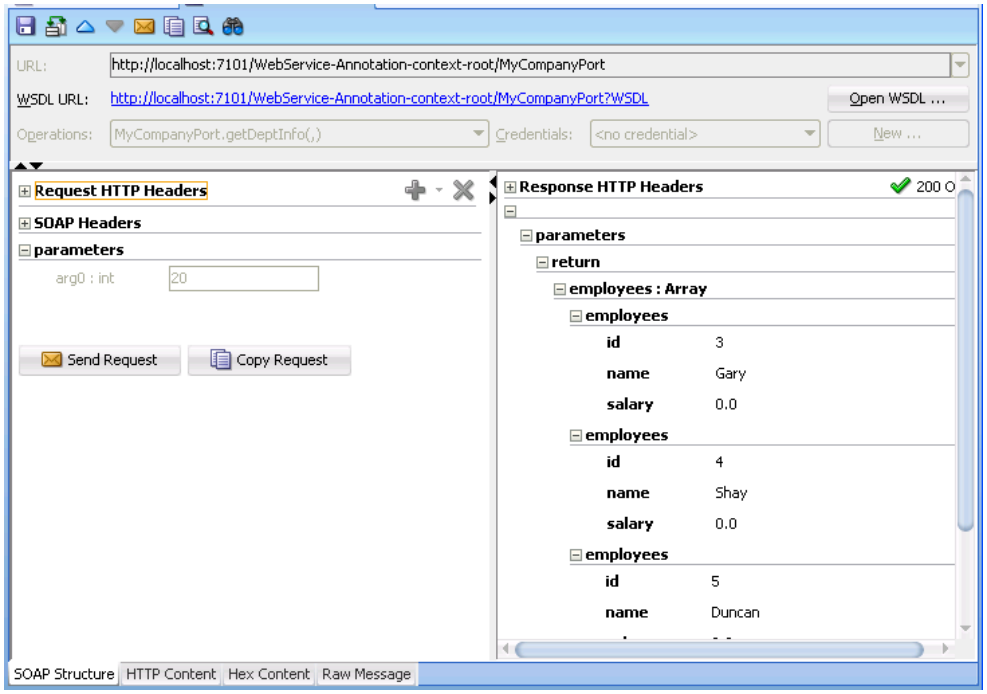
Icon	Name	Function
	Send Request	Click to resend a request when you have changed the content of a request. The changed request is sent and you can see any changes in the response that is returned.
	Open WS-I log file	Click to open the Select WS-I Log File to Upload dialog, where you can navigate to an existing WS-I log file that you create by saving the output of the HTTP Analyzer. For more information about WS-I, see the web site of The Web Services-Interoperability Organization (WS-I) at `http://www.ws-i.org`.
	Save Packet Data	Click to save the contents of the HTTP Analyzer Log Window to a file. For example, when you want to run the HTTP Analyzer in simulator mode, you can create a tape rule that references a WS-I log file (extension `.log`) that you save and revise. See Using the Tape Rule.
	WS-I Analyze	This feature is de-supported. Previously, running the WS-I Analyze wizard required installing the WS-I Testing Tools extension from the JDeveloper Update Center and that extension has been removed in JDeveloper 12c.
	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.

How to Use the Test Window

Use Test Web Service from the context menu of a web service container in the Applications window to see details of a request or response in a test window. See messages in the SOAP format or as HTTP code.

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 Applications window, or when you double-click a line in the HTTP Analyzer Log Window, illustrated in the figure below. By default, its position is in the center of JDeveloper, in the same place that the source editor appears.

Figure 11-5 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 the figure above.
- 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
```

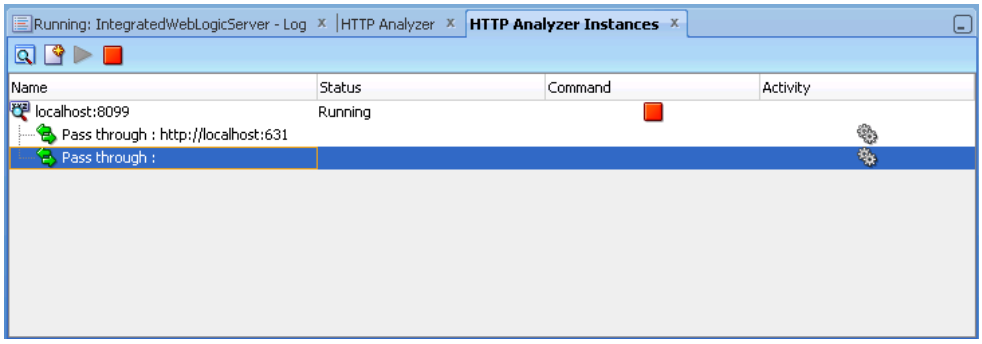
```
<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ns1="http://annotation/">
    <env:Header/>
    <env:Body>
        <ns1:getDeptInfo>
            <arg0/>
        </ns1:getDeptInfo>
    </env:Body>
</env:Envelope>
```

How to Use the Instances Window

Open the HTTP Analyzer Instances window using HTTP Analyzer from the Tools menu to view instances of the HTTP Analyzer that are running, or had been running. The window allows you to start and stop an instance.

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 11-6 HTTP Analyzer Instances Window



You create a new instance in the HTTP Analyzer page of the Preferences dialog, which opens when you click .

Table 11-9 HTTP Analyzer Instances Window Toolbar Icons

Icon	Name	Function
	Analyzer Preferences	Click to open the HTTP Analyzer page of the Preferences 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.

Table 11-9 (Cont.) HTTP Analyzer Instances Window Toolbar Icons

Icon	Name	Function
	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.

What Happens When You Run the HTTP Analyzer

Run the HTTP Analyzer to send the traffic of all Java processes and application server activity with JDeveloper by means of the HTTP Analyzer, using the proxy settings in the HTTP Analyzer page of the Preferences dialog.

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 page of the Preferences dialog, which opens when you click the Start HTTP Analyzer button in the Instance or Log window. 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.

How to Specify HTTP Analyzer Settings

Specify HTTP analyzer settings such as a different host and port number. Use the procedure to set HTTP Analyzer preferences.

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**.

How to Use Multiple Instances

When multiple instances of the HTTP Analyzer run, each instance uses a different host and port combination, and the HTTP Analyzer Instances window displays instance summaries. Use the procedure to add an additional HTTP Analyzer Instance.

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.

How to Configure External Web Browsers

Use external web browsers to route messages through the HTTP Analyzer to view traffic between a web browser and a client. Use the procedure to configure a Firefox profile for the HTTP Analyzer.

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.

Using SSL with the HTTP Analyzer

JDeveloper secures the HTTP Analyzer with the preconfigured HTTPS Credential, the HTTPS Keystore to encrypt HTTP messages, username tokens for basic authentication, PKI X509 certificates for SSO, Secure Token Service to manage security tokens over HTTPS, the OAuth protocol for authentication, and `jps-config.xml` to access JRF web services.

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.

When you run the service or application the analyzer uses the supplied credentials for perform the appropriate action.

The HTTP Analyzer can use the following types of credential:

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 information about keystores and keystore providers, see *Configuring Keystores*. When the default credential HTTPS Keystore 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.
You can create extra HTTPS keystores in addition to the default one provided by JDeveloper.

Username Token
Username token is a way of carrying basic authentication information. You supply a username/password to provide authentication.

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.

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.

OAuth Details
OAuth is an open authentication protocol that allows users to approve application to act on their behalf without sharing their password. You can test resources protected by OAuth protocol security using a valid OAuth credential, for example, to test resources that use Twitter feeds.

> **Note:**
>
> To access resources owned by users with a service provider you must first register with the service provider.

Import from JPS Config
Use to test JRF web services. `jps-config.xml` contains the security information that the HTTP Analyzer needs to access the service.
You can import of data from `jps-config.xml` into a credential record. You provide the location of `jps-config.xml` and enter the name of a CSF key and JDeveloper creates a credential record with the necessary data read from `jps-config.xml` and from the underlying wallet file. Data imported from `jps-config.xml` is populated in the X509 Certificates tab and in the Username Token tab.
Once saved, the credential record can be used repeatedly in the HTTP Analyzer and during web service proxy generation as the source for the keystore, keys and other required data.

How to Use SSL with the HTTP Analyzer

JDeveloper comes with a set of HTTPS Keystore credentials called `HTTPS Credential`. You can:

- Configure the client keystore for `HTTPS Credential` if mutual authentication is required.
- Create a new credential of one of the supported types.
- Create a new credential based on an existing credential.
- Delete a credential. Note that you cannot delete `HTTPS Credential`.

To configure credentials:

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. Make the appropriate changes:
   - Add client information for `HTTPS Credential` by choosing the HTTPS Keystore tab.
   - Add a new credential by clicking and selecting the appropriate tab.

- Create a new credential based on an existing credential by clicking . A new credential is created of the same type and with the same values with a default name of `Credential_n` incremented by 1.
- Delete a credential by selecting it from the list and clicking . Note that you cannot delete `HTTPS Credential`.

How to Run the HTTP Analyzer

JDeveloper provides a set of HTTPS Keystore credentials called HTTPS Credential to configure a client keystore, create, and delete credentials. Use the procedure to configure credentials.

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 Applications window.

How to Debug Web Pages Using the HTTP Analyzer

Using the HTTP Analyzer to debug Web pages allows you a direct view of the to-and-fro web traffic. Use the procedure 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.

How to Edit and Resend HTTP Requests

If you need to resend a HTTP request, edit the message and verify the changes in the message response. Use the procedures to send a request, and to edit and resend a request.

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:

- 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.

How to Use Rules to Determine Behavior

Set rules such as Pass Through Rule, Forward Rule, URL Substitution Rule, and Tape Rule so that the rules determine how the HTTP Analyzer runs.

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

Using the Pass Through Rule

The Pass Through rule passes a request to the service if the URL filter provides a match. Define **Pass Through Rules** in the Rule Settings dialog which displays two default rules.

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.

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.

Using the URL Substitution Rule

Use the URL Substitution rule to replace parts of URL ranges so that you can re-host services.

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.

Using the Tape Rule

Run the HTTP Analyzer with the tape rule in the simulator mode using the standard WS-I log file as the input to the rule. Use the procedure to test a web service client that is developed to run against an external web service.

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.

How to Set Rules

Set rules to determine HTTP Analyzer functioning such as redirecting requests to a different host or URL, or emulating a web service. Use the procedure to set rules for an HTTP Analyzer instance.

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 Applications window.

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.

Using the HTTP Analyzer with Web Services

Use the HTTP Analyzer to examine the content of 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.

Testing Web Services with the HTTP Analyzer

Test web services using the HTTP Analyzer to examine the network traffic of a proxy that is connected to a web service and was developed in JDeveloper. Use the procedure to test a web service.

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 Applications window, 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 Servers window.
   - 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.

Using the HTTP Analyzer with RESTful Web Services

The HTTP Analyzer reads a WADL created by Jersey for the RESTful web service, and the WADL is displayed in the HTTP Analyzer Test window. Use the procedure to examine the RESTful service, to test the RESTful service, and to work with the resource.

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

Java API for RESTful Web Services (JAX-RS) provides support for creating Web services according to the REST architectural style. JAX-RS uses annotations to simplify the development of RESTful Web services. By adding annotations to your Web service, you can define the resources and the actions that can be performed on those resources. WebLogic Server supports Jersey 2.x (JAX-RS 2.0 RI). Information about Jersey 2.x is available at: https://jersey.java.net/download.html.

JAX-RS 2.0 API Specification (Rev a) is available at: https://jax-rs-spec.java.net/nonav/2.0-rev-a/apidocs/index.html.

A Web Application Description Language (WADL) is an XML file created by Jersey that provides a description of the resources in the servlet. 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.

The following example provides an example of a WADL document which uses `POST`, `GET` and `DELETE`.

```
  <?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

Testing 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 Applications window, 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 Integrated WebLogic Server node in the Application Servers window.
   - 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.
6. To view the WADL file at anytime, click **Open WADL** to the right of the RESTful web service URL. A read-only version of the WADL file opens in the source editor, enabling you to view the a summary of the RESTful web service resources and methods, or the WADL source. .

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.

Using the HTTP Analyzer with WebSockets

The HTTP Analyzer passes unsecured WebSockets requests by means of a proxy. The HTTP Analyzer displays the content of the request response stream.

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`.

Using the HTTP Analyze with Fast Infoset

If you want the HTTP Analyzer to work with the Fast Info set, override the Content-Type in the HTTP headers, so that it does not lapse into working with the default soap/xml.

The HTTP Analyzer works with Fast Info set, but it will default to sending soap/xml instead unless you override the `Content-Type` in the HTTP headers.

- **SOAP 1.1** Change the `Content-Type` to `application/fastinfoset`.
- **SOAP 1.2** Include the action name, for example

  ```
  application/soap+fastinfoset;action=&quot;http://project1/HelloWorldSOAP12/
  helloRequest
  ```

Reference: Troubleshooting the HTTP Analyzer

This section contains information to help resolve problems that you may have when running the HTTP Analyzer.

Running the HTTP Analyzer While Another Application is Running

Stop any application that is waiting for a response before you start or stop the HTTP Analyzer.

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.

Changing Proxy Settings

Change the proxy settings in JDeveloper when you use the HTTP Analyzer. Use the procedure to set the HTTP proxy server, and edit the exception list.

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 Applications window, 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 and 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.

12

Running and Debugging Java Projects

This chapter describes how to use the tools and features provided by JDeveloper to run and debug Java programs. 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 object or several of them on the same or different machine, because JDeveloper supports distributed debugging.
JDeveloper provides several debugging windows (for example, Breakpoint, Heap, and Stack) that enable you to identify problem areas in your code. In addition, the various JDeveloper debugger and runner icons available from areas in the JDeveloper user interface are described.

This chapter includes the following sections:

- About Running and Debugging Java Programs
- Understanding the Processes Window
- Configuring a Project for Running
- How to Run a Project or File
- Macros
- Setting the Classpath for Programs
- Debugging Java Programs
- Using the Debugger Windows
- Managing Breakpoints
- Examining Program State in Debugger Windows
- Debugging Remote Java Programs

About Running and Debugging Java Programs

JDeveloper tracks run, debugged, or profiled Java processes, and supports local and remote debugging. For local debugging, set breakpoints in source files to start a session before starting the debugger, and for remote debugging, launch the debuggee process to which the debugger process is attached.

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.

JDeveloper supports two types of debugging: 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.

Understanding the Processes Window

The Processes window opens to track processes that are run, debugged, or profiled. Use Processes in Window to open a Processes window, which when a process has completed, automatically removes it from the window.

The Processes window keeps track of processes that are run, debugged, or profiled. When two or more such processes are active at the same time, the Processes window is automatically displayed. When a process has completed, it is automatically removed from the Processes window.

- To open the Processes window, choose **Window** > **Processes** from the main menu.
- To terminate a process within the Processes window, right-click a process in the Processes window and choose **Terminate** from the context menu.
- To view the Run Log, right-click a process in the Processes window and choose **View Log** from the context menu.

Configuring a Project for Running

Run configurations group the settings that control the running of programs. A project may use several run configurations, either custom or those that are bound to a project.

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.

> **Note:**
>
> Java programs that are run from JDeveloper, for example the Oracle ADF Model Tester, do not inherit the JDeveloper IDE Java options. Therefore in most cases you should set the run/debug Java options you want to use in the run configuration.

How to Choose a Run Configuration

A new project has a default run configuration created for it, but you can create a custom run configuration. Use the procedure to select a run configuration.

A default run configuration is created for each new project. You can select it or any other configurations you have created to run a selected project.

To choose a run configuration:

1. From the main menu choose **Application** > **Project Properties**.
2. Select **Run/Debug**.

> **Note:**
>
> If you have not previously accessed the Run/Debug/Profile page, a button labelled **Launch** displays. Click the button to access the **Run/Debug/Profile** page.

3. In the **Run Configurations** list, choose a run configuration.

> **Note:**
>
> The last edited run configuration is the active run configuration.

How to Create a Run Configuration

Use the procedure to create a run configuration. Copy an existing configuration to create a new run configuration, and modify its settings.

You create a new run configuration by copying an existing one, for example, Default. Then you modify the settings for the new configuration

To create a run configuration:

1. Select a run configuration as described in How to Choose a Run Configuration.
2. Click **New**.
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**.
6. To create a new run configuration having different settings, choose the new settings and click **OK**

How to Run a Project or File

Compile a project or file, run it, and view warning or error messages in the log window. Use the procedure to run a project or file.

After compiling your project or file, you can run it. For information about building a project or file, see Building Java Projects The log window displays any warning or error messages that may occur during the run.

To run a project or file:

1. In the Applications window, 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** > ***<target>***.
   - Right click and from the context menu, select **Run**.
   - Click the **Run** icon on the toolbar.

- Press the F11 key.

The main method of your Java application starts.

How to Run a Project from the Command Line

A project can be run from the operating system command line if it is a standalone executable, and the class file containing the application main() method is selected.

In order to run a project from the operating system command line:

- The project must be 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.

How to Change the Java Virtual Machine

The Java Virtual Machine may need changes according to operating systems: client-side applications need the HotSpot Client VM, whereas executing long-running server applications need the Server VM. Use the procedure 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 Applications window and choose **Project Properties** from the context menu.
2. Open the Run/Debug/Profile page.
3. Select a run configuration and click **Edit**.

   The Edit Run Configuration dialog displays.
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.

Macros

Macros are a series of commands and actions that can be stored and run whenever you need to automate tasks that are performed repeatedly and regularly.

A macro helps you to automate a task that you perform repeatedly or on a regular basis. It is a series of commands and actions that can be stored and run whenever you need to perform the

task. However in our case, the macro facility allows you to use a placeholder to represent an expression whose value is not known at the time the macro is being written.

Path locations that vary by installation or platform are a very common use for macros. Some examples might help clarify:

- Built-in IDE macros (qualifier = ide) include oracle.home, workspace.dir, and project.dir
- Environment variables (qualifier = env) provide access to platform-specific environment variables; they include JAVA_HOME, WAS_HOME, and HOSTNAME
- System properties (qualifier = prop) provide access to the Java system properties such as java.version, java.io.tmpdir, and file.separator
- User-Defined properties can be anything the user wants to define where none of the predefined macro handlers provide the needed value.

MacroHandlers are the classes that map the macro names to macro values. The JDEV-runner module provides you the options to define new macros, select macros for use in expressions, edit macros, etc.

Setting the Classpath for Programs

A Java program run from the command line needs the JVM with a list of the class paths to the files and libraries that comprise the application. The method that is used to run the Java program determines the form of the classpath.

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.

Setting the CLASSPATH Environment Variable (for java.exe)

The Java2 Standard Edition includes the `java.exe` file is a development tool that can be used to test a Java application from the CLI.

The `java.exe` file 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.

Using the JDeveloper Library CLASSPATH

JDeveloper ships hundreds of library classes from J2SE, third-party developers, and Oracle Corporation to help you generate your Java programs.

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.

> **Note:**
>
> Never use quotation marks in the classpath even when there is a space character in one of the paths.

Setting the CLASSPATH to Include Your Projects

If the output path is set to the default directory, test the Java application using `java.exe` by appending the classes directory to your classpath.

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.

Setting the CLASSPATH Parameter (for java.exe)

The Java Runtime Engine (`java.exe`) does not use the `CLASSPATH` environment variable, so the complete `CLASSPATH` to your Java program, or `classpath`, must be included as a parameter to the `java.exe` command.

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. Quotation marks are optional if there are no spaces in any of the `CLASSPATH` directory names.

Debugging Java Programs

The local and remote Debugger provides a number of features to investigate code, identify, and fix problem areas.

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.

Table 12-1 contains the special-purpose debugging windows that enable you to efficiently identify the problematic areas in your code.

Table 12-1 Debugging Windows

Window	Description
Breakpoint	Displays the breakpoints for the current workspace and project. See How to Use the Breakpoints Window.
Smart Data	Displays the data which is being used in the code that you are stepping through. See How to Use the Smart Data Window.
Data	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. See How to Use the Data Window.
Watches	Displays the watches created for fields or variables. 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. See How to Use the Watches Window.
Inspector	Displays a single data item in its own floating window. An inspector evaluates an expression according to the current context. SeeHow to Use the Inspector Window.
Heap	Displays information about the heap in the program you are debugging and helps you to detect memory leaks in your program. See How to Use the Heap Window.
Stack	Displays the call stack for the current thread. See Using the Stack Window.

Table 12-1 (Cont.) Debugging Windows

Window	Description
Classes	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. SeeHow to Use the Classes Window.
Monitors	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. SeeHow to Use the Monitors Window.
Threads	Displays the threads and the thread groups, highlights the current thread, and shows the name, status, priority, and group of each thread. See How to Use the Threads Window .

Understanding the Debugger Icons

The JDeveloper user interface, including the Debugger window and the Log window, provides several debugger and runner features.

Table 12-2 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 12-2 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.

Table 12-2 (Cont.) Debugger and Runner icons

Icon	Name	Description
	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 Processes window, 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 Processes window.
	Delete Breakpoint	Represents Delete Breakpoint in the toolbar. Click to remove a breakpoint.
	Disable	Represents a disabled breakpoint in the source editor margin and a disabled breakpoint in the Breakpoints window. The icon also represents the Disable Breakpoint command in the toolbar. Click to disable a breakpoint.
	Disable All Breakpoints	Represents Suspend All Breakpoints command in the toolbar. Click to run the application without stopping at the breakpoints, then return to debugging mode.
	Edit Breakpoint	Represents the Edit Breakpoint command in the toolbar, which you can use to edit the selected breakpoint.
	Enabled	Represents an active breakpoint in the source editor margin and the Enable Breakpoint command in the toolbar. Click to enable the selected breakpoint. When the debugger is not running, the enabled breakpoint icon looks same as the unverified breakpoint icon.
	Garbage Collection	Represents the **Run** > **Garbage Collection** menu option and the Garbage Collection command in the toolbar, 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.

Table 12-2 (Cont.) Debugger and Runner icons

Icon	Name	Description
	Resume	Represents the **Run** > **Resume** menu option and the Resume toolbar button which you can click.
	Run	Represents a running process in the Processes window, 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 **Window** > **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 (Shift + F7)	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.
	Threads	Represents the **Window** > **Debugger** > **Threads** menu option and the tab icon for the Threads window.
	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
	Verified Breakpoint	Represents the verified breakpoint in the source editor margin.

Debugging an Application Deployed to Integrated WebLogic Server

When debugging an application deployed to an integrated WebLogic Server, the changes you make to files are reflected in the running application without having to stop and redeploy the application.

When debugging an application deployed to an integrated WebLogic Server, you can make changes to files and they will be reflected in the running application without you having to stop and redeploy the application. To debug the application, start the server in debug mode. Go to *ORACLE_HOME*`/jdeveloper/jdev/bin` and run `java -?` to see the run configuration options available for the version of the JDK you are running.

- Modify JSF files

Save and refresh the browser if you are currently viewing the page that you modified, or navigate to the page if you were not on it. The page will reflect the changes you just made.

- Save and recompile them. You can see the changes the next time the altered code is executed without having to re-start the application.

If you are stopped at a break point, you can:

- Modify the class you are currently stopped in including the method that you are currently stopped in.
- Right-click in the editor and rebuild the individual class.
- Select the Stack tab (by default grouped with the Structure tab under the Applications window). At the top of the stack you will see that the method you are stopped in is obsoleted.
- Right click on the previous method and select the Pop back to here menu item. JDev will re-position the cursor to the code that invoked the obsolete method.

When you restart execution, the changes you made to the class or method will be reflected.

> **Note:**
>
> you can only modify code that is private to the class. Review the console output after re-compiling. The output will make it clear whether of not the recompiled code is available for run-time.

How to Debug a Project in JDeveloper

Compile code with debugging information, and use debugger features such as viewing arguments and local variables in the Data window. Use the procedure to set breakpoints and step through the code.

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. This saves time when debugging and testing an application.

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.

   The unverified breakpoints icon appears in the left margin.

2. Select **Run** > **Debug *filename.jpr***.

   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.
5. Double-click a method to display the associated source code in the source editor.
6. In the Smart Data and Data windows, examine the arguments and variables.
7. Display the Threads window to see the status of other threads in your program.

How to Edit and Recompile

While debugging, use the HotSwap feature to fix errors and replace class definitions, without stopping and restarting the application. HotSwap is an enhancement to the Java Platform Debugger Architecture that is implemented in the JVM in JDeveloper.

While debugging, use the HotSwap feature to fix errors and substitute corrected class definitions, without stopping and restarting your application. HotSwap is an enhancement to the Java Platform Debugger Architecture (JPDA) in Java 2 SDK v1.4, and has been implemented in JDeveloper's Java virtual machine, OJVM.

When the debugger is paused you can recompile classes. When the debugger resumes after code has been HotSwapped, threads that are currently executing old method definitions will continue to do so until those methods return. For subsequent method calls, the class definition will be used. However, existing instances of the class will not be modified, and class variables will not be reset. Use HotSwap to modify the logic of class methods. In most cases you cannot use HotSwap to make more substantial changes, such as adding or removing fields or methods.

Using FastSwap Deployment to Minimize Redeployment

Changes made to a deployed Web application are expected to be displayed when a browser is refreshed. IDEs provide incremental compilation support for editing and build steps. On the server side, the WebLogic Server FastSwap deployment feature in JDeveloper makes for smooth deployment and test cycles.

Today's Web application developers expect to make changes to a deployed application and see those changes immediately by refreshing the browser. On the Java EE side, developers typically have to go through the following cycle to see their changes in action.

```
Edit -> Build -> Deploy -> Test
```

These steps, along with the many required descriptor elements, makes developing applications with Java EE seem complex and top-heavy. Among these steps, the build and deploy cycles are necessitated by Java and by the application server being employed. IDEs are trying to make the edit and build steps seamless by providing incremental compilation support. On the server side, the WebLogic Server FastSwap deployment feature, available in JDeveloper, makes the deploy and test cycles just as seamless.

How FastSwap Deployment Works

The Java EE dynamic class redefinition is limited in that the declared fields and methods of a class cannot change. The purpose of FastSwap alleviates this restriction in WebLogic Server by allowing the dynamic redefinition of classes with new shapes to facilitate iterative development.

Java EE 5 introduced the ability to redefine a class at run time without dropping its classloader or abandoning existing instances. This allowed containers to reload altered classes without disturbing running applications, vastly speeding up iterative development cycles and improving the overall development and testing experiences. The usefulness of the Java EE dynamic class redefinition is severely curtailed, however, by the restriction that the shape of the class – its declared fields and methods – cannot change. The purpose of FastSwap is to remove this restriction in WebLogic Server, allowing the dynamic redefinition of classes with new shapes to facilitate iterative development.

With FastSwap, Java classes are redefined in-place without reloading the classloader, thereby having the decided advantage of fast turnaround times. This means that you do not have to wait for an application to redeploy and then navigate back to wherever you were in the Web page flow. Instead, you can make your changes, auto compile, and then see the effects immediately.

Supported FastSwap Application Configurations

A FastSwap deployment requires that the WebLogic Server runs in the development mode, and that support is available for changes to class files in exploded directories.

The following application configurations are supported when using FastSwap deployment:

- FastSwap is only supported when WebLogic Server is running in development mode. It is automatically disabled in production mode.
- Only changes to class files in exploded directories are supported. Modifications to class files in archived applications, as well as archived JAR files appearing in the application's classpath are not supported. Examples are as follows:
  - When a Web application is deployed as an archived WAR within an EAR, modifications to any of the classes are not picked up by the FastSwap agent.
  - Within an exploded Web application, modifications to Java classes are only supported in the `WEB-INF/classes` directory; the FastSwap agent does not pick up changes to archived JARs residing in `WEB-INF/lib`.

Enable FastSwap in Your Application

Enable FastSwap in an application by creating a deployment descriptor for the Oracle WebLogic server in the `weblogic-application.xml` file.

To enable FastSwap in your application, create an Oracle WebLogic deployment descriptor `weblogic-application.xml` and enable Fast Swap.

FastSwap can also be enabled for a standalone Web application by creating the WebLogic deployment descriptor`weblogic.xml` and enabling Fast Swap.

Overview of the FastSwap Process

The steps describe how the FastSwap deployment process works.

See Limitations When Using FastSwap.

The following steps describe how the FastSwap deployment process works:

1. Once FastSwap is enabled at the descriptor level, an appropriate classloader is instantiated when the application is deployed to WebLogic Server.
2. Open a browser to see the application at work. Modify (add/edit/delete) the methods and/or classes and then compile them.

   Note that the FastSwap agent does not compile Java files.
3. Refresh the browser or send a new request to the application.

   The FastSwap agent tries to find all classes that have been modified since the last iteration by looking at all directories in the classpath. Considering an exploded application with a single Web application, the following directories are examined for any class file modifications based on their timestamps:

```
ExampleApp/APP-INF/classes
ExampleApp/webapp/WEB-INF/classes
```

The FastSwap agent redefines the modified classes in the application and then serves the request.

Application Types and Changes Supported with FastSwap

JDeveloper supports FastSwap with POJOs, web applications, and enterprise applications deployed in the exploded format although no support is available for resource adapters.

FastSwap is supported with POJOs (JARs), Web applications (WARs) and enterprise applications (EARs) deployed in an exploded format. FastSwap is not supported with resource adapters (RARs).

The following types of changes are supported with FastSwap:

- Addition of static methods
- Removal of static methods
- Addition of instance methods
- Removal of instance methods
- Changes to static method bodies
- Changes to instance method bodies
- Addition of static fields
- Removal of static fields
- Addition of instance fields
- Removal of instance fields

The following table lists detailed change types supported with FastSwap:

Table 12-3 Supported Application Types and Changes

Scope	Java Change Type	Supported	Notes
Java Class	Add method	Yes	Addition of the `finalize` method is not supported.
Instance (non-abstract)	Remove method	Yes	Addition of the `finalize` method is not supported.
	a) Add field	Yes	
	b) Remove field	Yes	
	c) Change method body	Yes	
	d) Add constructor	Yes	
	e) Remove constructor	Yes	
	f) Change field modifiers	Yes	
	g) Change method modifiers	Yes	
Class-level (static)	Add method	Yes	
	Remove method	Yes	
	Change body method	Yes	

Table 12-3 (Cont.) Supported Application Types and Changes

Scope	Java Change Type	Supported	Notes
Class Hierarchy Changes	Change list of implemented interfaces	No	
	Change extends "SuperClass"	No	
Abstract Java Class	Add abstract method	Yes	
	Delete abstract method	Yes	
	All other supported changes (a–g) listed in `Instance`	Yes	
"final" Java Class	Same supported changes (a–g) listed in `Instance`	Yes	
"final" Java Method	Same supported changes (a–g) listed in `Instance`	Yes	
"final" Java Field	Same supported changes (a–g) listed in `Instance`	Yes	
Enum	Add constants	No	
	Remove constants	No	
	Add/remove methods	No	
Anonymous Inner Class	Add/remove fields	NA	Not supported by the Java language
	Add/remove methods	No	
Static Inner Class	Same supported changes (a–g) listed in `Instance`	Yes	
Member Inner Classes (non-static inner classes)	Same supported changes (a–g) listed in `Instance`	Yes	
Local Inner Classes	Same supported changes (a–g) listed in `Instance`	Yes	
Java Interface	Add method	Yes	
Java Reflection	Access existing fields/methods	Yes	
	Access new methods	No	New methods are not seen using Reflection and some synthetic methods are exposed.
	Access new fields	No	New fields are not seen using Reflection.
Annotations on Classes	Add or remove method/field annotations	No	
Annotation Type	Add or remove methods/attributes	No	
Exception Classes	Same supported changes (a–g) listed in `Instance`	Yes	
EJB Interface	Add/remove methods	No	Changes to EJB interfaces involve Reflection, which is not fully supported.
EJB 3.0 Session/MDB EJB Implementation Class	Add/remove methods	No	Any support classes referenced by the EJB classes can be modified.
	Add/remove fields	No	
EJB 3.0 EntityBean	Add/remove methods	No	Any support classes referenced by the EJB classes can be modified.

Table 12-3 (Cont.) Supported Application Types and Changes

Scope	Java Change Type	Supported	Notes
	Add/remove fields	No	
EJB Interceptors	Add/remove methods	No	Any support classes referenced by the EJB classes can be modified.
	Add/remove fields	No	

Limitations When Using FastSwap

Several limitations, such as the unavailability of support for the addition or removal of Java annotations or methods on EJB Interfaces, are associated with FastSwap deployment.

The following limitations apply when using FastSwap deployment:

- Java reflection results do not include newly added fields and methods and include removed fields and methods. As a result, use of the reflection API on the modified classes can result in undesired behavior.
- Changing the hierarchy of an already existing class is not supported by FastSwap. For example, either a) changing the list of implemented interfaces of a class; or b) changing the superclass of a class, is not supported.
- Addition or removal of Java annotations is not supported by FastSwap, since this is tied to the reflection changes mentioned above.
- Addition or removal of methods on EJB Interfaces is not supported by FastSwap since an EJB Compilation step is required to reflect the changes at run time.
- Addition or removal of constants from Enums is not supported.
- Addition or removal of the finalize method is not supported.
- When you change a field name, the object state is not retained. This type of change occurs as follows: the field with the old name is deleted and a field with the new name is added. As such, any state in the old field is not carried over to the renamed field. You should expect an instance value to be reset when you change a field name.

Handling Unsupported FastSwap Changes

When FastSwap is enabled, it redefines classes in the existing classloaders. If the redefinition fails, the JVM throws an `UnsupportedOperationException` both in the Log window and in the server log file while the application continues to run.

When FastSwap is enabled, after you recompile a class, FastSwap attempts to redefine classes in existing classloaders. If redefinition fails because your changes fall outside the scope of supported FastSwap changes, the JVM throws an `UnsupportedOperationException` in the Log window and in the server log file. Your application will not reflect the changes, but will continue to run.

To implement your changes, you can redeploy the application or affected modules (partial redeploy), depending on the application type and the extent of your changes.

How to Debug ADF Components

JDeveloper supports ADF Declarative Debugger so that applications can be debugged with breakpoints, and breakpoint types decide how the application moves from one breakpoint to another. An application execution pauses to facilitate the comparison of the data that is available, and the expected data for the functioning of the binding container.

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.

Table 12-4 contains the windows that JDeveloper provides for debugging ADF components.

Table 12-4 ADF Component Debugging Windows

Window	Description
ADF Data	Displays relevant data based on the selection in the ADF Structure window when the application is paused at a breakpoint. For more information, see How to Use the Data Window.
EL Evaluator	Evaluates EL Expressions when a breakpoint is reached during a debugging session. Only JSF applications can use the EL Evaluator.
ADF Structure	Displays a tree structure of the ADF runtime objects and their relationships when the application is stopped at a breakpoint. For more information, see 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. See How to Export Debug Information to a File.

How to Use JDeveloper Debugger to Execute a Program

Use the procedure to use the JDeveloper debugger to control the execution of a program. If the fix proves effective, exit the debugging session, fix the code, and recompile to save the changes.

To use the JDeveloper debugger to control the execution of a program:

1. Run to a breakpoint.

   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.

   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..

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.

   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.

   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.

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.

How to Configure a Project for Debugging

Manage the way a program is debugged, how it enables and disables packages and classes, and configures remote debugging options. Use the procedure to configure debugger and remote debugger options in JDeveloper.

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)
2. Choose **Application** > **Project Properties** (to set preferences that apply only to the current project).
3. Select the **Run/Debug** node.
4. Select a run configuration.
5. Click **Edit**.
6. Select **Tool Settings** > **Debugger**.
7. Set the options on the Debugger and Remote pages.
8. Click **OK** when finished.

How to Set the Debugger Start Options

Set the debugger start option to specify how the debugger functions at the start of a new debugging session. Use the procedure 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** > **Debugger**.

2. Select a Start Debugging option.

Table 12-5 Start Debugging Options

Option	Description
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.

Note:

You can press F7 to display a choice of methods you want to step into. When there are multiple methods, they are highlighted when you press F7. You can click the one you want to step into.

This feature is useful there can be multiple method calls on a line, and you may be interested in a specific one that might not be the first one encountered.

How to Launch the Debugger

Specify build options on the Compiler page of the Project Properties, and build the project because the debugger starts if the project builds successfully. Use the procedure to build a project and start the debugger.

You must build the project before debugging it. The project is built using options you specify on the Compiler page of the Project Properties.

To build a project and start the debugger:

1. In the Applications window, select the project.
2. Right-click and choose **Project Properties** > **Compiler**.
3. If not already enabled, select **Full Debug Info**.
4. Click **OK** to close the dialog.
5. 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** ***<target>***.
   - 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.

How to Export Debug Information to a File

From a debugger window, export the debug information that the JDeveloper debugger generates to a text or HTML output file. Use the procedure 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 a 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.
5. Click **OK** to close the Preferences dialog.
6. In the debugger window, right-click and choose **Export**.
7. 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).

Using the Source Editor When Debugging

When the debugger stops, the source editor displays the source file for the current open class marked by an icon for execution.

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 the figure.

Figure 12-1 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 to:

- Hover over a variable name to see its value.
- Set a breakpoint, click in the source editor's margin.
- Remove a breakpoint, click the breakpoint in the source editor's margin.

Figure 12-2 Breakpoint Icon

Using Context Menu Items

The debugger adds the Run to Cursor, Watch, Inspect, and Step Into Method at Cursor menu items to the context menu of the source editor.

The debugger adds several menu items to the source editor's context menu, including those shown in Table 12-6.

Table 12-6 Debugger Context Menu Items

Item	Function
Run to Cursor	Run to the current location of the cursor and execute the code until it reaches that location.
Watch (Ctrl+F5)	Add an expression to the Watches Window.
Inspect	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 displays tooltips 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 that 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.

Using Java Expressions in the Debugger

Specify the legal Java expressions that are used in the Watches window, Inspector window, Breakpoint Conditions, and Breakpoint Log Expressions for the debugger to monitor and evaluate.

Java expressions are used in the Watches window, Inspector window, Breakpoint Conditions, and Breakpoint Log Expressions. You can specify a Java expression that you want the debugger to watch or inspect. The expression must be a legal Java expression that the debugger can evaluate. The debugger accepts Java expressions in the forms shown in Table 12-7.

Table 12-7 Java Expressions Accepted by Debugger

Java Expression	Form
Simple variable name	`rect`
Field access	`rect.width`
Method call	`myString.length()`
Array element	`myArray[3]`

Table 12-7 (Cont.) Java Expressions Accepted by Debugger

Java Expression	Form
Array length	`myArray.length`
Comparison operation	`rect.height == 100 myArray.length > 7`
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`

Moving Through Code While Debugging

Use the debugger to manage how JDeveloper executes a single line of code, a method, or a code block in a program. Use the Step Into and Step Over commands to traverse program code.

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. See How to Set the Debugger Start Options.

The debugger lets you control the execution of your program by:

- 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.

How to Step Into a Method

Set the debugger to start by stepping into code using the Step Into command, so that the program that is being debugged executes until a method in a tracing-enabled class is reached. Use any of the listed methods to step 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.

The **Step Into** action will highlight methods on a line any time it is invoked on a line with multiple methods. The current execution point determines what line of code the debugger will step into. Moving the cursor using the arrow or tab key alters the execution point for the method to step into.

This feature is particularly useful when a selected outer method belongs to an interface and you do not know beforehand where the method that will actually get called is implemented.

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 the F7 key.
- Click the **Step Into** button from the toolbar.

Figure 12-3 Step Into Button

> **Note:**
>
> **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 are not sure that a method is well behaved, step into the method and check whether it is working properly.

How to Step Over a Method

Set the debugger to start by using the Step Over command so that the program being debugged executes until a method in a tracing-enabled class is reached, but does not stop in the `class static initializer` method. Use any of the listed methods to step 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 the F8 key.
- Click the **Step Over** button on the toolbar.

Figure 12-4 Step Over Button

Unlike previous releases of JDeveloper, you cannot start debugging by clicking 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.

Controlling Which Classes Are Traced Into

Set the tracing include and exclude lists in Project Properties or Classes before debugging. The descending classes within the specified package are included or excluded.

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.

How to Step Through Behavior as Guided by Tracing Lists

If a class or package is excluded, the debugger that is set to step into a class runs through that code without pauses. Use the procedure to adjust the tracing include or exclude list.

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:

To adjust the tracing include or exclude list:

1. Right-click a project in the Applications window and choose **Project Properties** from the context menu.
2. Select the **Run/Debug** node.
3. Choose a run configuration and click **Edit**.
4. In the Edit Run Configuration dialog select **Tools Settings** > **Debugger**.
5. In the **Tracing Classes and Packages to Include** and **Tracing Classes and Packages to Exclude** fields, enter the name of the packages or classes you want to include or exclude in the appropriate field, separated by a semicolon (;). For example:

   ```
   oracle.xml;org.apache;org.omg;org.w3c;org.xml
   ```

   You can also 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
```

How to Locate the Execution Point for a Thread

During the debugging process, the line of code that is the current execution point for the thread is highlighted and the left margin of the source editor displays the icon for the execution point. Use the procedure to find the current execution point.

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.

How to Run to the Cursor Location

Step through application code in the debugger to a particular location without single stepping or setting a breakpoint. Use the procedure to run to a specific program 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 the F4 key. The F4 key works unless you Operating System (OS) intercepts that key.

> **Note:**
>
> On ADC or SLC machines, F4 key is intercepted and never reaches the debugger.

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.

How to Pause and Resume the Debugger

Pause debugger running at a point in the program so that the debugger can evaluate the code at the location, and then resume debugging. Use the Pause option for Run, or use Pause on the debugger toolbar to pause 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 12-5 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 displays, 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**.

How to Terminate a Debugging Session

Terminate a debugging session using Terminate in the Run menu if the debugger has run past a bug location, and needs a restart. Termination of a session closes all debugger windows without deleting breakpoints or watches that had been set.

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 - *<target>*** 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.

How to View the Debugger Log

Select Log for Debugger in the Window menu to view the Debugger log for an active debugger session.

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:

- Select **Window > Debugger** > **Log**
- In the Processes window, right-click the process and select **View Log** in the context menu.

Using the Debugger Windows

Use special-purpose debugging windows to analyze code and identify problem areas. Use the options that are available by either right-clicking in a window and selecting Preferences from

the context menu, or by selecting Preferences for Tools, and saving the debug information as text or HTML output file.

JDeveloper provides a number of special-purpose debugging windows to help you analyze your code and identify problem areas. 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.

How to Open Debugger Windows

Set a breakpoint to start a debugging session and at a breakpoint, select a specific window from the Debugger menu for Window. You can open and use Debugger windows for classes, Heap, Data, Log, Monitors, Stack, Smart Data, Threads, and Watches depending on requirements.

You open a Debugger window by setting a breakpoint and starting a debugging session. When the Debugger stops at the breakpoint, select **Window** > **Debugger** > *deubugger_window*. For example, **Window** > **Debugger** > **Threads**. You can access the Breakpoints windows by simply starting a debugging session (it is also accessible from **Window** > **Breakpoints**)

Select one of the following Debugger windows depending on the information you want to see: Classes, Heap, Data, Log, Monitors, Stack, Smart Data, Threads, and Watches. See the following sections for information on using these windows.

How to Use the Breakpoints Window

Select Breakpoints from the Window menu to open a window to display a list of set breakpoints. The Breakpoints window displays information about set breakpoints.

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 display a list of set breakpoints:

- Choose **Window** > **Breakpoints** from the main menu. The Breakpoints window appears.

How to Use the Data Window

The Data window displays information about variables in a program. It also displays the current return value of a non-void method when a breakpoint in the method is set, and the Step to End of Method command or Step Out command is run.

You use the Data window to display information about variables in your program. In 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.

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.

How to View Array Elements in the Data Window

Set the number of elements in an array to be displayed in the Data Window. Use the procedure to view array elements in the Data window.

You can set the number of elements in an array that you want to display in the Data Window.

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.

How to Use the Smart Data Window

The Smart Data window displays data that is relevant to the source code that is being stepped through. Set the options for smart data on the Smart Data page, selecting Debugger for Preferences for Tools.

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. See 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.

How to Use the Watches Window

A watch monitors the changing values of variables or expressions as a program runs. The values of the variables in the watch expression change to reflect program updates and changing contexts.

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 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.

How to Add a Watch from the Source Editor

Use the procedure to add a watch from the Source Editor. A watch can also be added by selecting Watch from the context menu of an item selected in the Data window or by dropping items from the Data window to the Watches window.

You can add a watch to an expression in the source editor.

To add a watch from the Source Editor:

1. In the source editor, 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**.

You can also add a watch in the following ways:

- 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.

How to Watch a Static Field

To view a static field, enter the full name of the class followed by a period and the name of the field.

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
```

How to Edit a Watch

Edit a watch by selecting its expression in the Watches window. Use the procedure to edit a watch.

You can edit a watch by selecting its expression in 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**.

How to Delete a Watch

Delete a watch by selecting its expression in the Watches window. Use the procedure to delete a watch.

You can edit a watch by selecting its expression in the Watches window.

To delete a watch:

1. Select the expression in the Watches window.
2. 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.

How to Use the Inspector Window

In the Inspector window, select a variable, field, or object, and get the information from the Watch or Data windows displayed. Use the procedure to open the Inspector Window.

The Inspector window enables 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. See Using 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.

How to Use the Heap Window

The Heap window displays information about the heap in the program that is being debugged, and helps identify memory leaks in a program. The heap window holds the Class Folder and the Reference Path Folder.

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 use the Heap window:

- 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.

Using the Stack Window

The Stack window displays the call stack for the current thread.

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.

How to Use the Stack Window

In the Stack window, you can highlight a line in the stack thread to update values in the Data, Watches, and Inspector windows.

How to View the Stack of a Tread

The Stack window displays the stack of a selected thread. Use the procedure to view the stack of a thread.

You can view the stack of a selected thread in the Stack window.

To view the stack of a thread:

1. Start debugging the project and open the Stack window.
2. Select the thread from the dropdown list, above the columns.

The Stack window immediately reflects the stack of the selected thread.

How to Use the Classes Window

The Classes window displays the classes that have been loaded and information such as the number of instances of a class and the memory needed. The window also shows garbage collector runs.

The Classes window displays which classes have been loaded and may also include useful information, such as the number of instances of a class and how much memory that number of instances requires. 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.

How to Change the View Order

You can change the order in which items in each column of the Classes Window displays.

To change the ascending or descending view order:

- Click at the top of a 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 display 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 12-6 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 12-6 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

How to Use the Monitors Window

Java uses the synchronization mechanism to coordinate activities and data access among multiple threads. The Monitors window displays status and control information for active monitors.

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.

The Monitors window will also open automatically if the debugger is in a deadlock state.

How to Use the Threads Window

The Threads window displays the names, status, and thread groups of all the threads in a program. Use Threads for Debugger in the Preferences for Tools to set the columns that get displayed in the window.

The Threads window displays the names and status of all the threads and thread groups in your program. The columns that display in this window depend on those column settings that are enabled in the **Tools** > **Preferences** > **Debugger** > **Threads** page or by choosing Preferences from the Threads window context menu options which you can access by right-clicking in the Threads window

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.

How to Set Preferences for the Debugger Windows

You can customize the settings for the Debugger window. Use the procedure to set preferences for the Debugger window.

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 debugger (the program you are debugging), try increasing the connection retry setting.

To set any of the Debugger window preferences:

1. Choose **Tools** > **Preferences** > **Debugger**.

   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.

How to Specify Which Columns Display in the Window

The debugger windows only displays the columns that you select. Right-click in the window and select Preferences from the context menu, or right-click a column heading to specify which columns get displayed in the window.

You can choose which columns display in each of the debugger windows. *

To specify which columns display in the window:

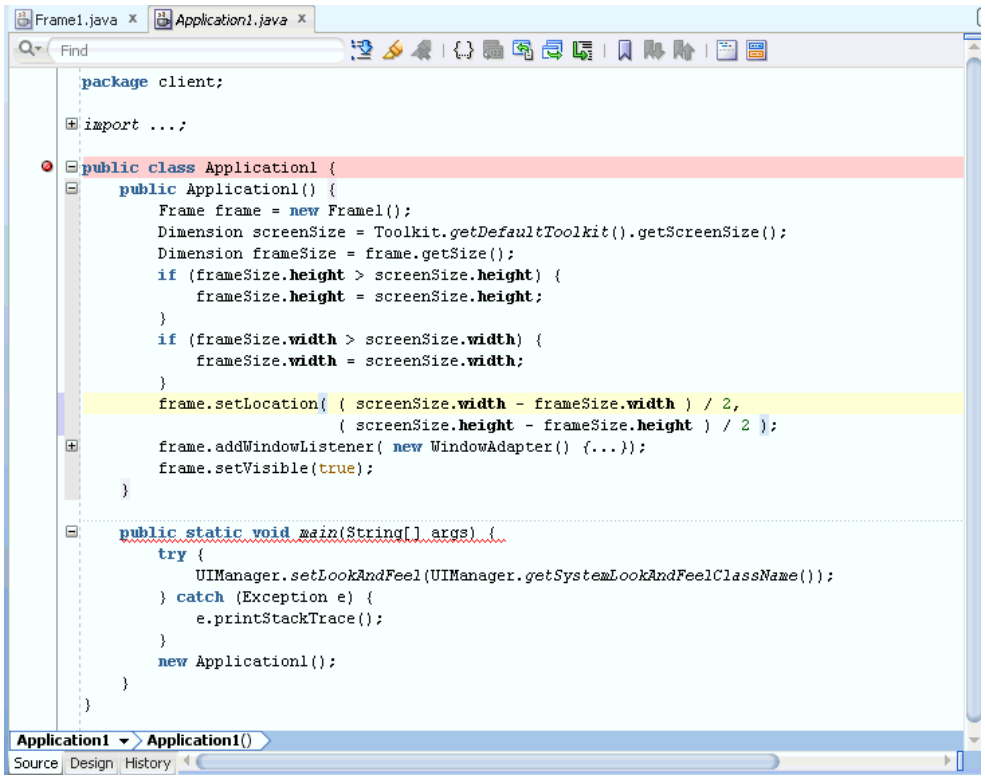
- Right-click in the window and choose **Preferences** from the context menu. Under **Columns**, select the columns you want to be displayed in the Breakpoints window.
- Or, in the window, right-click on the columns heading and select column names.

Managing Breakpoints

Set breakpoints in source code so that the running of the program pauses at points that needs debugging. JDeveloper supports Source breakpoints, Exception breakpoints, Method breakpoints, Class breakpoints, File breakpoints, and Deadlock 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. The figure displays an example breakpoint in a Java Application source file.

Figure 12-7 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.

Understanding Verified and Unverified Breakpoints

During debugging, the debugger verifies all breakpoints by checking if they have been set in executable lines of code, and displays an icon to indicate that they have been validated.

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 the figure.

Figure 12-8 Verified Breakpoint Icon

Understanding Deadlocks

A deadlock, such as the common monitor block cycle deadlock, blocks one or more threads in a program from gaining access to a resource, or waits on a condition that cannot be satisfied. To debug, pause your program by clicking Pause in the debugger, and open the Monitors window.

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.

The following example shows a typical Java synchronization deadlock:

```
synchronized (a)
 {
```

```
...
synchronized (b)  {
...
 }
 ...
}
```

At the same time, thread 2 is executing the following code:

```
synchronized (b)
{
...
  synchronized (a)
  {
  ...
   }
   ...
 }
```

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.

Understanding the Deadlock Breakpoint

You can disable the persistent deadlock breakpoint that the debugger automatically sets when JDeveloper starts. The debugger pauses program running when it encounters a monitor block cycle deadlock.

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.

Understanding Grouped Breakpoints

JDeveloper supports Grouped breakpoints that allow you to use multiple breakpoints as a set. Enable breakpoints that are disabled when the debugger encounters the set points for program pauses, or set source breakpoints so that the debugger automatically enables the breakpoint group which contains the exception breakpoint.

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.

How to Edit Breakpoint Options

Using the Edit Breakpoint dialog, edit the options for a breakpoint after it has been added in the source code. Use the procedure to view and modify the options of a breakpoint.

JDeveloper allows you to edit the options of a breakpoint after you have added it in the source code. 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.

To view and modify the options of a breakpoint:

1. If the Breakpoints window is not open, select **View** > **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.

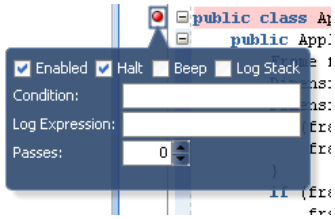
4. Make any necessary changes to the breakpoint options.
5. To accept the changes, click **OK**.

Editing a Breakpoint

Right-click a breakpoint located in the source editor to edit its attributes, such as enabled, disabled, or condition, in the Edit Breakpoint dialog.

You can right-click to edit a breakpoint located in the source editor. The Edit Breakpoint dialog shown in the figure displays, where you can specify the definition of the breakpoint. In the dialog, you can edit some of the most important breakpoint attributes, such as enabled/disabled, condition and more.

- **Figure 12-9 Edit Breakpoints Dialog**



You can also hover over a breakpoint in the source editor to display the dialog in the figure above.

How to Set Source Breakpoints

A source breakpoint that is set on a line of executable code in the source pauses program execution. Set a source breakpoint by using Breakpoints in Window, or Toggle Breakpoint after right-clicking in the left margin next to a line of code.

A source breakpoint is the most common type of breakpoint. It is set on a line of the source code and program execution is paused when it hits that line.

To set a source breakpoint, do one of the following:

- 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 **Window** > **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 Breakpoint** 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 might 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.

How to Control Breakpoint Behavior

Manage debugger functioning at a breakpoint by setting halt execution, beep, the enabling or disabling of a group of breakpoints. Use the procedure to control how the debugger behaves when a breakpoint occurs.

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

How to Delete a Breakpoint

Delete a breakpoint when it is not required using either the source editor or the Breakpoints window. Use the detailed options to delete a breakpoint.

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.

To delete a breakpoint, do one of the following:

- 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.

How to Disable a Breakpoint

Disabling a breakpoint retains all the defined breakpoint settings, but does not trigger the breakpoint during the program run. Disable a breakpoint using the options that are detailed.

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 a breakpoint, do one of the following:

- In the source editor, right-click the breakpoint symbol in the left margin and choose **Disable Breakpoint**.
- In the Breakpoints window (**Window** > **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.

How to Enable a Breakpoint

You can re-enable a disabled breakpoint by using the Enable, Enable All, and the Enable Group options from the right-click context menu in the Breakpoints window. You can also use Enable on the Breakpoints toolbar.

You can reenable a disabled breakpoint.

To reenable a disabled breakpoint:

- 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.

How to Set Instance Breakpoints

Instance filters that are displayed in the Instance Filters column of the Breakpoints window are source breakpoints with instance filters that identify selected instances. Use the procedure to set an instance breakpoint.

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.
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 want to track.
4. Resume the debugger.

   The debugger will stop at the instance breakpoint only for the selected instances.

How to Set Exception Breakpoints

Set exception breakpoints that are activated when a certain type of exception is thrown during the running of a program. Use the procedure to set an exception breakpoint.

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.

How to Make a Breakpoint Conditional

The debugger pauses program execution every time a breakpoint is encountered, and by setting a condition using the Edit Breakpoints dialog, you can customize breakpoints so that they are activated only in certain conditions. Use the procedure to set a breakpoint condition.

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 a breakpoint condition cannot be evaluated, the debugger will stop at the location as if it was an unconditional breakpoint.

When the debugger stops at a line with a conditional breakpoint, a ghost window appears next to the breakpoint icon showing the breakpoint condition. If the debugger stopped because the condition was true, the condition is shown with a green checkmark icon; if it stopped because the condition could not be evaluated, the condition will be shown with a question mark icon.

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. Or, with your mouse cursor, hover over a breakpoint icon in the gutter of an editor window.

Using Pass Count Breakpoints

Use the Pass Count field to specify the number of times that a breakpoint must be passed before a breakpoint is activated. If the default number, one, is set, when the program reaches the breakpoint line, the breakpoint is activated, and the program pauses.

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.

How to Examine Breakpoints with the Breakpoints Window

Select Breakpoints in the Debugger menu for Window to view a list of breakpoints, and the breakpoints that have been validated by the debugger. Use the procedure to use the Breakpoints window to locate a breakpoint in the source editor.

To see the list of breakpoints, choose **Window** > **Debugger** > **Breakpoints** from the main menu. Breakpoints that have been verified as valid by the debugger are indicated by the icon shown in Figure 12-8. 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.

How to Manage Breakpoint Groups

Create a breakpoint group, enable or disable it, or remove it like a single breakpoint.

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.

How to Create a Breakpoint Group

Create a breakpoint, and edit it to create a breakpoint group so that all breakpoints in the selected group can be enabled collectively. Use the procedure to create a breakpoint group.

You create a breakpoint group by first creating a breakpoint, then editing it to create a breakpoint group. This enables all the breakpoints contained in the selected group. When a breakpoint is enabled, it means that the conditions and actions are executed when the breakpoint is encountered.

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.

How to Move a Breakpoint into a Breakpoint Group

Use the procedure to move a breakpoint into a breakpoint group, or drag-and-drop the breakpoint into the breakpoint group.

To move a breakpoint, you can either drag-and-drop it into the breakpoint group, or follow these steps.

To move a breakpoint into 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. 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.

Enabling Disabling or Removing a Breakpoint Group

In the Breakpoints window, right-click a breakpoints group, and select the Enable Group, Disable Group, or Delete Group options from the context menu to enable, disable, or remove a breakpoint 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.

Examining Program State in Debugger Windows

Program variables reveal bugs. Pause a program in the debugger to examine the values of variables, arguments, fields, and array items.

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.

How to Inspect and Modify Data Elements

When debugging, use the Data, Smart Data, Inspector, or Watches windows to view and edit data items. In the Source Editor, when the program pauses at a breakpoint, hover the mouse pointer over an item to get its details.

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.

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.

How to Inspect a Data Item

When debugging, test a data item with different expressions to spot bugs, and if an edit fixes an error, exit the program, fix the bug, and then recompile the program. Use the procedure to inspect a data item.

When you inspect a data item, you evaluate it with different expressions while your debugging session is running. 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.

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.
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.

How to Modify the Value of a Variable

When debugging, test hypothetical bug fixes, and if a fix fits, exit the program, fix, and then recompile, but change program source code as the fix does not change the code. Use the procedure to modify the value of a variable in the Data window.

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.

> **Note:**
>
> Some object types cannot be modified while the program is running. A warning appears if you attempt to modify such an object type.

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 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.

How to Modify Expressions in the Inspector Window

Use the inspector window to modify an existing expression. Use the procedure to modify an expression in the Inspector window.

You can modify an existing expression in the inspector window.

To modify an expression in the Inspector window:

1. You can type the new expression in the corresponding text box, or 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**.

How to Show and Hide Fields in the Filtered Classes List

When debugging, in a data-related debugger window, filter the number of displayed fields for an expanded object so as to narrow focus, locate, and isolate potential errors in the program. Use the procedure to show or 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 **Types window** 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.

Debugging Remote Java Programs

Debug code on a remote computers or on VM instances. Manually launch the program to be debugged, and use the remote debugging protocols on the Debugger - Remote page of the Edit Run Configuration dialog to debug code that has been deployed remotely.

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. See Auditing and Monitoring Java Projects.

- Select **Attach to JPDA** to attach to the debugger application at a specified address. For information about the Java Platform Debugger Architecture (JPDA) Connection and Invocation, see `http://docs.oracle.com/javase/6/docs/technotes/guides/jpda/conninv.html`
- Select **Listen for JPDA** 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.

How to Start a Java Process in Debug Mode

To Start a Java process in the Debug mode, configure the project for remote debugging, start the remote debugging session, and run the command appropriate to the debugging protocol and the environment.

After you have 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.

> **Note:**
>
> The options shown are applicable if you are running JDK 1.6 or later. See the documentation for the version of JDK you are running if earlier than JDK 1.6.

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 information about the JPDA Connection and Invocation, see http://docs.oracle.com/javase/7/docs/technotes/guides/jpda/conninv.html.

How to Use a Project Configured for Remote Debugging

A project must be configured for remote debugging. Use the procedure to configure a project 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. T

   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.

How to Configure JPDA Remote Debugging

Configure JDeveloper for remote debugging by using the Java Platform Debugger Architecture. Use the procedures to configure a project for remote debugging, and to configure JDeveloper for remote debugging.

This section describes how to 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.

13

Implementing Java Swing User Interfaces

This chapter describes how to create graphical user interfaces (GUIs) for applications using the Swing components.
Using the Swing GUI builder in JDeveloper, you can quickly and easily assemble the elements of a user interface (UI) for a Java application using Swing components. You construct the UI with components selected from the Components window, 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.

Included is a description of the UI debugger, which is used to debug user interfaces specifically for AWT and Swing-based client applications. The UI Debugger offers an alternative way of debugging a GUI application.

This chapter includes the following sections:

- About Java Swing UI Components and Containers
- Designing Java GUIs
- How to Create a Form
- Understanding the Forms You Can Create
- Working with Layout Managers
- Adding Components
- Working with Containers
- Working with Layout Managers
- How to Create Accessible Forms
- Working with Event Handling
- How to Modify GUI Source Code
- Working with the UI Debugger

About Applications Developed in Earlier Versions

Migrate applications that were developed using Swing in earlier versions of JDeveloper, and open them in the current version of JDeveloper. JDeveloper supports working with previously designed visual classes.

Applications which use Swing which were developed in earlier versions of JDeveloper can be migrated and opened in this version of JDeveloper, and there is full support for working with previously designed visual classes. The legacy editor works with any class that has a `jbInit()` method.

You can visually edit forms in the old application using the old Java Visual Editor, however the way it worked and the way the new Swing GUI builder works are different, and there is no migration from one to another.

About Java Swing UI Components and Containers

Java Swing components, such as buttons, tables, text components, and split panes are used in designing application UI. JDeveloper provides tools to generate containers, components that hold and manage other components, that are used in designing the UI, and a layout manager automatically arranges the components in a container.

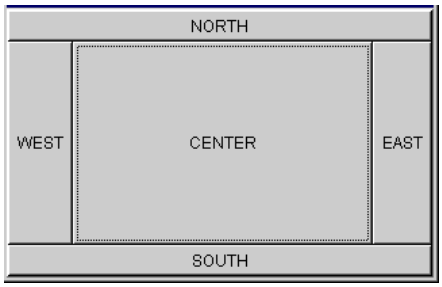
Java Swing components include everything from buttons, tables, text components, and split panes. For example, the `JCheckbox` component is a square box used to display Boolean (true/false) values. Many components are capable of sorting, printing, and drag and drop, as well as other supported features.

You can see examples of Java Swing in the tutorials, which are available at `https://docs.oracle.com/en/java/javase/17/docs/api/java.desktop/javax/swing/package-summary.html`.

When you lay out a form to design the UI in JDeveloper, you use a container. Containers are also components. They hold and manage other components. You interact with them by getting and setting their properties, calling their methods, and responding to their events as with any other component. JDeveloper provides tools to help you generate your containers. See Working with Containers for more information.

A layout manager automatically arranges the components in a container according to a particular set of rules specific to that layout manager. It determines the size and position of the components within a container. For example, the `BorderLayout` arranges a container's components in areas named North, West, East, South, and Center.

Figure 13-1 BorderLayout



Designing Java GUIs

Use the GUI Builder to create and modify application GUI in Design view for which JDeveloper automatically generates the Java code in Source view to implement the interface. GUI components are added to a form using the Components window, and modified using the Properties window.

The IDE's GUI Builder enables you to design GUI's (graphical user interfaces) visually. As you create and modify your GUI, the IDE automatically generates the Java code to implement the interface.

Whenever you open a GUI form, the IDE displays it with tabs enabling you to switch between Source and Design views. The Design view enables you to work with GUI forms visually while the Source view permits editing of the form's source code directly.

Typically, you add components to a form using the Components window and arrange them in the GUI Builder workspace. As you work, the GUI Builder automatically displays guidelines suggesting preferred alignment and anchoring for the components you add. Using the Properties window in conjunction with the Structure window, you can then examine and adjust properties of a form's components and layout managers, manage component event handlers, and define how code is generated.

Notes:

- GroupLayout was added to version 6 of the Java Platform. You can set the version of GroupLayout in the property sheet for each form.
- To use the GUI Builder, you must work with files that were created with the IDE's GUI form templates. You cannot use the GUI Builder to edit GUI forms (that is Java files) that were created outside of the IDE.

About Guarded Blocks

When designing the UI with forms in the Design view, the GUI Builder generates uneditable code called guarded code which it displays in the Source view. Guarded code includes blocks of variable declarations of components, the `initComponents()` method, in which form initialization is performed, and the headers of all event handlers.

As you work with a form in the Design view, code is generated automatically by the GUI Builder and is displayed in the Source view with a grey background. This code is called guarded code and is not directly editable.

> **Note:**
>
> You can freely edit all the code that is not guarded. For example, you can write code for initialization or customization of the UI (in the constructor just after the `initComponents()` call).
>
> All the components have assigned fields so they can be accessed from anywhere in the Java class.

Guarded text generated by the GUI Builder's includes:

- Blocks of components' variable declarations.
- The `initComponents()` method, in which form initialization is performed. This method is called from the form's constructor and though it is not editable manually, you can affect the way it is generated by editing the Code properties in the component's property sheet.
- The header (and trailing brace) of all event handlers.

How to Create a Form

Within a project, components are added to forms to create application UI. Use the procedure to create a new form.

You can create forms within an existing project.

To create a new form:

1. In the Applications window, select the project where you want to create the form.

2. From the main menu, select **New** > **From Gallery** > **Client Tier** > **Swing/AWT**.
3. Select the type of form you want to create.
4. Complete the wizard and click **OK**.
5. Scroll to the source file for the form in the Applications window, for example, `NewPanel.java`.
6. Double-click the source file.

   The GUI builder displays where you can edit the form.

Understanding the Forms You Can Create

JDeveloper provides forms to build Swing panels, frames, dialogs, internal frames, and AWT panels, forms, and dialogs.

The following table contains the forms you can build in JDeveloper.

Form	Description
JPanel Form	Creates a new Swing panel. Panels are frequently used to group together several other components into one place. A JPanel itself can be used as a component within a JFrame or JDialog.
JFrame Form	Creates a new Swing frame. Frames are typically used as stand-alone top level windows as the main user interface to the application. Most Swing applications are built starting from this form.
JDialog Form	Creates a new Swing dialog. Dialogs are modal or modeless windows that are typically used to prompt users for input.
JInternalFrame Form	Creates a new Swing internal frame that can be placed into a JDesktopPane to provide an MDI user interface.
Bean Form	Creates a new form based on any JavaBeans component (either visual or non-visual). You must specify the bean class in the wizard.
Panel Form	Creates a new AWT (Abstract Window Toolkit) panel. Panels are frequently used to group together several other components into one place. AWT is the toolkit used in previous versions of JDeveloper. Swing is preferred now.
Frame Form	Creates a new AWT form. Frames are typically used as stand-alone top level windows as the main user interface to the application.
Dialog Form	Creates a new AWT dialog. Dialogs are modal or modeless windows that are typically used to prompt users for input

Adding Components

There is more than one way to add components for display and control to a new form. Use the procedures to add a component from the Components window, add multiple components from the Components window, and add a component using the context menu.

Once you have created a new form, you can add components for display and control. You can add components a few different ways:

To add a component from the Components window:

1. Select a component in the Components window by clicking its icon.

2. Without releasing the mouse button, drag the component to the desired location in the form.
3. If you want to add a non-visual component, or a visual component, out of the visual hierarchy of the designed container, place it in the white area around the form. It will then appear under the Other Components node in the Structure window.

To add multiple components from the Components window:

1. Select a component in the Components window by clicking its icon. Release the mouse button.
2. While holding down the shift key, click each location in the form where you want to place an instance of the component you selected in the palette.
3. Release the shift key when adding the last component, or press Esc or right click to cancel adding.
4. If you want to add a non-visual component, or a visual component but out of the visual hierarchy of the designed container, place it in the white area around the form. It will then appear under the Other Components node in the Structure window.

To add a component using the context menu:

1. In the Structure window, right-click the container to which you want to add a component.
2. Choose **Add From Components** and then choose desired component from the appropriate submenu:
   - Containers
   - Controls
   - Menus
   - Windows
   - Box Fillers
   - AWT
   - Beans

How to Set Component Properties

Use the Properties window to modify the appearance and the functioning of a component that has been added to a form. Use the procedure to edit the properties of a component.

Once you have added a component to a form, you can adjust its behavior and appearance in the Properties window.

To edit a component's properties:

1. Select the component in the GUI Builder or Structure window to display its properties in the Properties window. Note that if you select multiple components, their common properties are displayed and any modifications apply to all of the selected components.
2. Edit the component's properties in the Properties window by selecting the property and entering the desired value.
3. If the property you want to edit has an ellipsis (...) button, you can click it to open a special property editor that provides more advanced editing options (e.g. a color chooser for a color-type property), or alternate ways of specifying the property value (e.g. using a resource bundle for a text property), including a possibility to type directly the code that should represent the property value.

Use this to edit a given property in another way than in the small in-place editor in the properties window. You can also choose from several ways how to enter the property value via the combo box at the top.

How to Select Components in Your User Interface

The selection arrow in the GUI builder toolbar must be depressed when selecting a component in the UI so as not to accidentally add a component to the UI.

Before attempting to select an existing component in your UI, be sure the selection arrow in the GUI builder toolbar is depressed. Otherwise, you may accidentally place a component on your UI.

How to Select a Single Components

Clicking the component in the GUI Builder, selecting the component in the Structure window, or using the designated keys selects a single component.

To select a single component, do one of the following:

- Click the component in the GUI Builder.
- With focus on the GUI Builder, tab to the component (Tab = forward; Shift+Tab = backward).
- Select the component in the Structure window

How to Select Multiple Components

Select multiple components using any of the listed methods.

To select multiple components, hold down the Ctrl key and do one of the following:

- Hold down the Ctrl key and click the components in the GUI builder or in the Structure window to add/remove components from selection one by one.
- Click and drag around the outside of the components you want to select. 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.

  If you need to drag for selection inside a sub-container, you would normally drag the sub-container away. To prevent that, press and hold Shift during dragging.
- Hold down shift holding Shift and click components in the Structure window to perform an interval selection. On design canvas clicking with Shift adds to selection one by one, but does not remove from selection.

How to Align Components

Adjust the alignment of added components so that the form appears as required at runtime. Use the procedure to align components.

Once you have added components, you can adjust their alignment them to ensure that your form will appear as desired at runtime.

Most of the time, you can achieve the desired alignments by "snapping" components at the suggested positions when dragging. Then you can further adjust the alignment using alignment actions from the toolbar or context menu.

> **Note:**
>
> This applies only to Free Design (the default layout mode), not to other layout managers.

To align components:

1. Select the components you want to align in the GUI Builder.
2. Click the appropriate align button in the GUI Builder toolbar.

   Alternately, you can right-click either component and choose **Align** > **Left in Column** (or **Align** > **Right in Column**) from the pop-up menu.

   The IDE shifts the component positions so that the specified edges are aligned and the component anchoring relationships are updated.

How to Size Components

Setting the same size to several related components in modal dialogs makes them easily recognized as offering similar functionality, and adjusting the resizing behavior of components ensures that component relationships are maintained at runtime.. Use the procedures to set components to the same size, and to set component resizing behavior.

It is often beneficial to set several related components, such as buttons in modal dialogs to be the same size so they can be easily recognized as offering similar functionality. You can also adjust the resizing behavior of components to ensure that component relationships are maintained at runtime.

The Free Design mode allows you to size components using mouse any way you need. Do this with consideration, though. Resizing a component from its default size to a fixed size leads to setting the component with a hardcoded size in pixels that may go against the cross-platform layout principles. Typically you do not want to resize buttons or labels, their size should only be defined by their text. It is usually fine to resize components with no fixed content that are set to "Auto Resizing" because they can still accommodate their size in runtime (for example, text fields).

When working with `null`, `AbsoluteLayout`, or `GridBagLayout`, you can size components when you first place them in your UI, or you can resize and move components later.

> **Note:**
>
> This applies only to Free Design (the default layout mode), not to other layout managers.

To set components to the same size:

1. Select all of the components you want to be the same size in the GUI Builder.
2. Right-click any one of components, and choose **Same Size** > **Set Width** (or **Same Size** > **Set Height**) from the context menu.

To set component resizing behavior:

1. Select the components whose auto resizing behavior you want to set.

2. Right-click any one of the components and choose **Auto Resizing** > **Horizontal** (or **Auto Resizing** > **Vertical**) from the context menu.

   Alternatively, use the toolbar buttons.

   Component auto-resizing behavior is set to resize horizontally at runtime. The alignment guidelines and anchoring indicators are updated to indicate the new component relationships.

Working with Containers

Java GUIs are forms comprised of grouped subcontainers within containers, various components that provide the required information, and control functionality. Use the procedures to change the GUI Builder focus to a specific container, and to return display focus of the GUI Builder to the entire form.

Java GUIs are forms comprised of top-level containers within which are grouped sub-containers as well as the various components used to provide the desired information and control functionality.

It is often useful to focus work on single subcontainers rather than the entire form the GUI Builder generally displays. When working with large forms containing complex nested hierarchies of containers, changing the scope of the GUI Builder's focus enables you to concentrate on specific parts of your interface.

To change the GUI Builder's focus to a specific container:

1. In the GUI Builder or Structure window, right-click the container you want to edit.
2. Choose **Design This Container** from the contextual menu.

   The IDE adjusts display of the workspace such that the current container fills the work area and hides the form's other components. The form's entire hierarchy remains available in the Structure window.

To return the GUI Builder's display focus to the entire form:

1. Right-click the container in the GUI Builder.
2. Choose **Design Parent > [Top Parent]** from the contextual menu.

   The IDE adjusts the work area display such that the entire form is visible. If the Design Parent menu item is dimmed, you are already designing the entire form.

Reordering Components Within a Container

The sequence in which components are added decide the sequence of the order of components in a container, and if the layout manager does not use constraints, the sequence also determines the visual arrangement of components. Set component order in the Structure window for Layout managers that use constraints.

The order of components in a container follows the sequence in which components are added. If the layout manager you have chosen for a container does not use constraints (FlowLayout, BoxLayout, and GridLayout), the order of components also determines how they are arranged visually. You can, however, reorder the components using the Structure window or by dragging them in the form itself.

With layout managers that use constraints (BorderLayout, GridBagLayout, CardLayout, AbsoluteLayout, and Null Layout), the order of components in the container does not determine the order in which the components appear. For these containers, you can only rearrange the component order in the Structure window. Although GridBagLayout uses

constraints to determine how components are arranged, component order determines the layout when the Grid X and Grid Y constraints are not used.

Working with Layout Managers

Instead of using `AbsoluteLayout` and null layout in production UI, which may not render a Java program on all platforms, use layout managers to specify the rules and constraints for the UI layout to make them portable. Change the layout manager using the Set Layout menu in the Structure window.

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. For this reason, you should not use AbsoluteLayout and null layout in production UI. These are not suitable for cross-platform UI, should be used only for prototyping or with awareness of their restrictions.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 enable you to control the way in which visual components are arranged in GUI forms by determining the size and position of components within containers. This is accomplished by implementing the LayoutManager interface.

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.

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.

The layout manager sets the sizes and locations of the components based on various factors such as:

- Layout manager's layout rules
- Layout manager's property settings, if any
- Certain properties common to all components, such as `preferredSize`, `minimumSize`, `maximumSize`, `alignmentX`, and `alignmentY`
- Size of the container

Normally, when coding your UI manually, you override the default layout manager before adding components to the container. To change a layout manager on the container, right click it in the designer area or in the Structure window, go to Set Layout menu and select the desired layout manager. See How to Set the Layout Manager.

When using the GUI builder (or visual editor), you can change the layout whenever you like. JDeveloper will adjust the code as needed.

> **Note:**
>
> If you want to change the properties for a layout manager using the GUI builder, you must explicitly specify a layout for a container so its properties will be accessible in the Properties window.

Choose a layout manager based on the overall design you want for the container. Some layouts can be difficult to work with in the GUI builder 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 layout called `null`, which leaves the components exactly where you place them and at the size you specify. Starting with 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.

If you cannot get what you need with the Free Design mode, experiment with different layouts to see their effect on the container's components. For example, a viable alternative of an complex layout is the GridBagLayout. See How to Use the GridBag Customizer. 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.

How to Set the Layout Manager

Set the default layout for a new container and use the Free Design features, or change the layout of containers using the GUI Builder or the Structure window. Use the procedures to set the layout manager from the GUI Builder, and to set the layout manager from the Structure window.

When you create a new container, it is generally created using a default layout so that you can take advantage of the IDE's Free Design features. If necessary, you can change the layout of most containers using the GUI Builder or the Structure window.

To set the layout manager from the GUI Builder:

1. Right-click the container whose layout you wish to change.
2. Select **Set Layout** and a layout menu

To set the layout manager from the Structure window:

1. Right-click the node for the container whose layout you wish to change.
2. In the contextual menu, choose the desired layout from the **Set Layout** submenu.

When you change layouts, the IDE remembers the properties of the discarded layout manager. If you then revert back the to the previous layout manager, the form also returns to its prior state.

Understanding FreeDesign Layout

Use FreeDesign for a form because it automatically suggests optimal alignment and spacing, and translates design decisions into a functional UI without requiring a layout manager. Free Design employs a dynamic layout model, and FreeDesign containers can be combined with containers using other layouts in the same form.

FreeDesign lays out your form using visual guidelines that automatically suggest optimal alignment and spacing. As you work, the GUI Builder translates your design decisions into a functional UI without requiring you to specify a layout manager. Because Free Design employs

a dynamic layout model, whenever you resize the form or switch locales, the GUI adjusts to accommodate your changes without changing the relationships between components.

- You can combine FreeDesign containers and containers using other layout managers together in the same form. Free Design enables you to lay out your form using visual guidelines that automatically suggest optimal alignment and spacing of components.

How to Set Layout Properties

Modify the appearance of forms by adjusting the manager properties of the general layout and properties that are specific to components. Use the procedures to set the manager properties of general layout , and to set layout properties of components.

You can modify the appearance of your forms by adjusting general layout manager properties as well as properties specific to components.

You can modify:

- General layout properties which affect all components in a container, such as alignment of components and gaps between the components.
- Layout properties specific to a component that is managed by a particular layout manager and which apply to that component alone. These type of properties are also known as constraints.

To set general layout manager properties:

1. Select the layout manager's node in the Structure window.
2. In the Properties window, select the property you want to edit and enter the desired value. Note that the properties vary depending on the layout manager and that some layout managers do not have any properties.

To set layout properties of components:

1. Select the component in the Structure window.
2. In the Properties window, scroll down to **Layout**, select the property you want to edit and enter the desired value.

> **Note:**
>
> You can edit the custom code of a component in a more natural way by selecting **Customize Code** from context menu of the component and then edit its custom code in more natural way.

Understanding Layouts Provided with JDeveloper

JDeveloper supports FlowLayout, BorderLayout, GridLayout, CardLayout, AbsoluteLayout, and Null Layout.

You can choose from the following Layout Managers in the IDE:

- FlowLayout

  FlowLayout arranges components in a container like words on a page. It fills the top line from left to right until no more components can fit, continuing the same way on each successive line below.

- BorderLayout

  BorderLayout arranges components along the edges or the middle of their container. Using BorderLayout, you can place components in five possible positions relative to the `ComponentOrientation` of the container:

  - First, which correspond to `BorderLayout.PAGE_START`
  - Last, which correspond to `BorderLayout.PAGE_END`
  - Before, which correspond to `BorderLayout.LINE_START`
  - After, which correspond to `BorderLayout.LINE_END`
  - Center, which corresponds to interior area.

- GridLayout

  GridLayout places components in a grid of equally sized cells, adding them to the grid from left to right and top to bottom.

- GridBagLayout

  GridBagLayout is a powerful layout manager that provides precise control over all aspects of the layout even when the container is resized, using a complex set of component properties called "constraints." It is particularly useful for multiplatform Java applications as it enables you to create a free-form layout that maintains a consistent appearance across platforms.

  GridBagLayout places components in a grid of rows and columns in which grid cells do not all have to be the same size. In addition, components can span multiple rows, columns, or both.

- CardLayout

  CardLayout provides a means of managing two or more components occupying the same display area. When using CardLayout each component is like a card in a deck, where all cards are the same size and only the top card is visible at any time. Since the components share the same display space, at design time you must select individual components using the Structure window.

- BoxLayout

  BoxLayout allows multiple components to be arranged either vertically or horizontally, but not both. Components managed by BoxLayout are arranged from left to right or top to bottom in the order they are added to the container. Components in BoxLayout do not wrap to a second row or column when more components are added or even when the container is resized.

- AbsoluteLayout

  AbsoluteLayout enables components to be placed exactly where you want them in the form, move them around in the IDE, and resize them using their selection borders. It is particularly useful for making prototypes since there are no formal limitations and you do not have to enter any property settings. However, it is not recommended for production applications since the fixed locations and sizes of components do not change with the environment.

- Null Layout

  The Null Layout is used to design forms without any layout manager at all. Like the AbsoluteLayout, it is useful for making quick prototypes but is not recommended for production applications, as the fixed locations and sizes of components do not change when the environment changes.

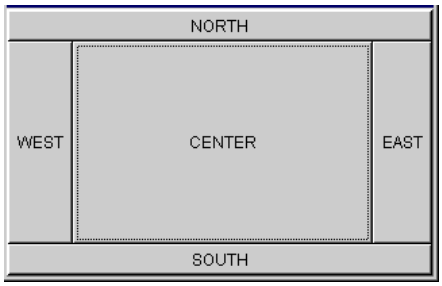
Using BorderLayout

`BorderLayout` arranges container components in areas named First, Last, Before, After, and Center. The properties of constraints determine in which area the components are positioned.

`BorderLayout` arranges a container's components in areas named First, Last, Before, After, Center.

- The components in First and Last are given their preferred height and are stretched across the full width of the container.
- The components in Before and After are given their preferred width and are stretched vertically to fill the space between the first and last areas.
- A component in the Center expands to fill all remaining space.

Figure 13-2 BorderLayout



The BorderLayout that appears in Figure 13-2 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 Properties window to one of the following values: First, Last, Before, After, 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 First area of the container. You do this by selecting the panel and choosing First for its constraints property in the Properties window.

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.
- If you need more components in one area of the `BorderLayout`, use the **Enclose In** context menu option to group the selected components into a sub-panel.

> **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 Properties window to specify the horizontal or vertical gap in pixels for a `BorderLayout` associated with a container.

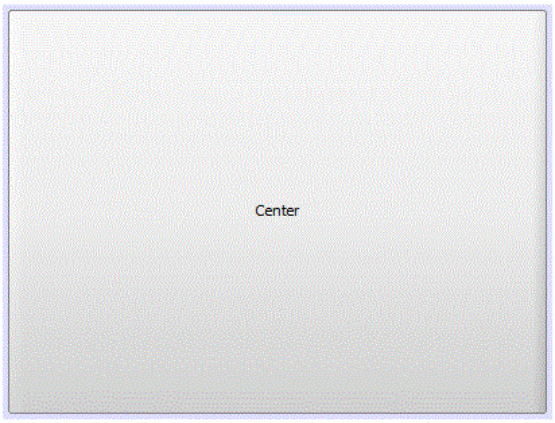
To modify the gap surrounding `BorderLayout` components, select the `BorderLayout` object in the Structure window (displayed immediately below the container it controls), then modify the pixel value in the Properties window for the horizontal gap and vertical gap properties.

Using CardLayout

`CardLayout` stacks components such as panels. If two or more panels need to share display space, use the `CardLayout` layout which allows an area to hold components that change periodically.

`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.

Figure 13-3 Card layout



`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.

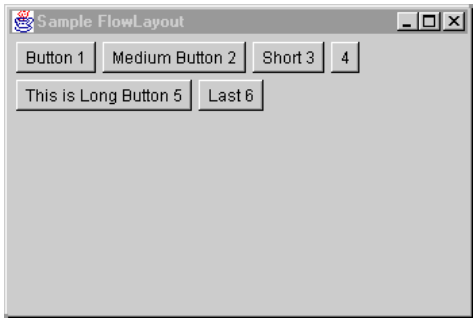
By selecting `CardLayout` in the Structure window you can then you can then specify the amount of horizontal and vertical gap surrounding stack of components in the Properties window.

Using FlowLayout

`FlowLayout`, the layout that is used to arrange buttons on a panel, organizes components in rows and columns by using the `preferredSize` of a component. Use the Properties window to change both the alignment and spacing properties of the layout in the GUI builder.

`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 13-4 FlowLayout



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 Properties window to change both the alignment and gap properties when you're in the GUI builder.

Changing the Alignment

To change the alignment, select the `FlowLayout` object in the Structure window, then specify a value in the Properties window for the alignment property.

Changing the Gap

Select the `FlowLayout` object in the Structure window, and modify the pixel value for the horizontal gap or vertical gap property in the Properties window to change the default `FlowLayout` of five pixels.

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 horizontal gap or vertical gap property in the Properties window.

Changing the Order of Components

Change the order of the components in a `FlowLayout` container by dragging a component to a new location, or by right-clicking a component and selecting Move Up or Move Down.

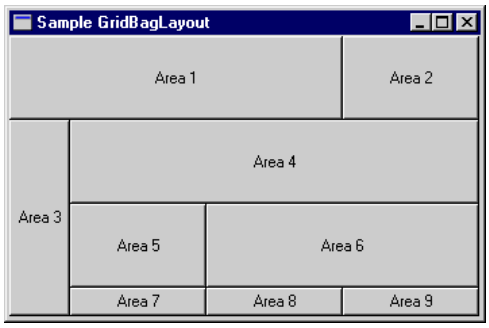
To change the order of the components in a FlowLayout container, drag the component to the new location, or right-click a component and choose **Move Up** or **Move Down**.

Using GridBagLayout

`GridBagLayout` is a flexible layout that positions components in rows and columns on a dynamic grid. Nesting components into smaller panels inside the `GridBagLayout` container offers greater control over components and uses fewer cells, but requires more memory for the program because it uses more containers.

`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 13-5 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.

Adding Components to a GridBagLayout Container

Components that are added to the design canvas appear in the default row. Reposition the components by using the `GridBag` customizer.

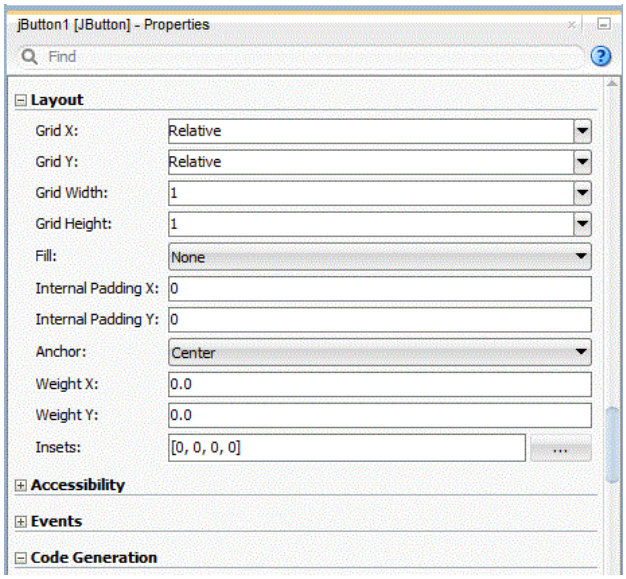
When you add components to the design canvas they appear in one row by default. You cannot position them using the mouse. Instead, you must use the GridBag customizer. See How to Use the GridBag Customizer.

How to Set GridBagConstraints in the Properties Window

If you want buttons in a `GridBagLayout` container to use the same internal padding, select each individually, and then edit the corresponding layout constraint property. Use the procedure to set layout properties in the Properties window.

Using the Properties window, you can specify some of the `GridBagConstraints`.

Figure 13-6 Layout Properties in Properties Window



If you want all the buttons in your `GridBagLayout` container to use the same internal padding, you can hold down the Ctrl key while you select each one, then edit the corresponding layout constraint property.

To set layout properties in the Properties window:

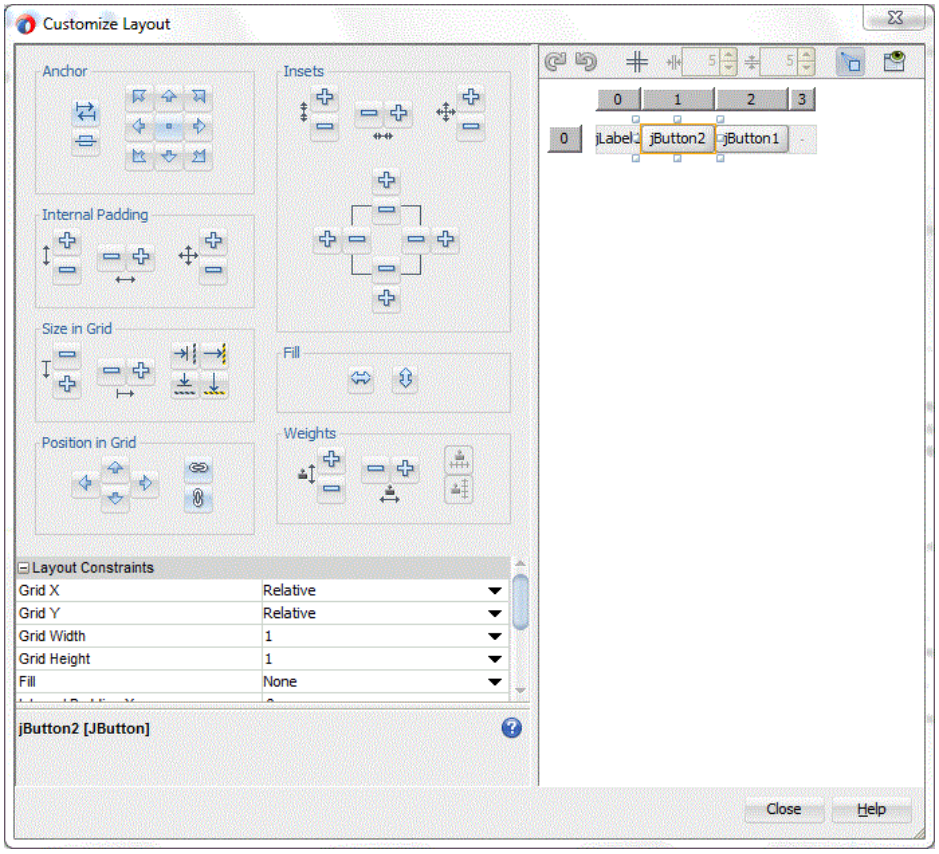
1. Select the component(s) within the `GridBagLayout` container you want to modify, either in the Structure window or in the GUI builder.
2. In the Properties window, select **Layout**.
3. Select a value for the constraints property in the Properties window.
4. Set the desired constraints in the property editor, then press **OK**.

How to Use the GridBag Customizer

The `GridBag` customizer enables you to visually adjust the placement and constraints of components in a `GridBagLayout`. Use the procedure to use the `GridBag` customizer.

The GridBag customizer enables you to visually adjust the placement and constraints of components in a GridBagLayout.

Figure 13-7 Customize Layout Dialog



It includes a property sheet for GridBag constraints, buttons for adjusting the constraints, and a rough depiction of the layout of the components. The GUI Builder more closely reflects how the components will look at runtime.

To use the GridBag customizer:

1. Add the components you require to your form and ensure the GridBagLayout is set for it.
2. To open the customizer, right-click the GridBagLayout node in the Structure window and choose **Customize** from the contextual menu.
3. Drag the components in the right pane to reposition them as desired.

   As you drag a component, its Grid X and Grid Y properties change to reflect its new position.
4. Once the approximate layout of the components has been established, select a component and adjust its constraints as desired in the left pane.

   Note that you can either enter the values directly or use the provided buttons to adjust the component's constraints.

   While editing:

   - You may need the Redo, Undo, Pad, and Test Layout buttons in the toolbar above the right pane.
   - You can right click the column/row headers and add or remove columns/rows.

- You can also right click an empty cell and add a new component (so you do not have to close the GridBag customizer dialog in order to access the palette).

If after several rows, your design has fit nicely into a certain number of columns, and you suddenly have a row that requires an odd number of components, then 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.

5. Once you are satisfied with the layout, click **Close** to exit the customizer.

Using GridLayout

Specify the number of rows and columns and the number of pixels between the cells in `GridLayout` components. Select the `GridLayout` object in the Structure window, and then edit the values in the Properties window to change the property values for a `GridLayout` container.

`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 13-8 GridLayout

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 and vertical gap 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, columns, horizontal gap and vertical gap properties in the Properties window.

Previewing a User Interface

Use Preview Design on the GUI Builder toolbar for a preview of the GUI when the program is compiled and run. The previewed GUI form delivers active components.

To quickly test how your GUI will display when it is compiled and run, click the **Preview Design** button in the GUI Builder toolbar. A dialog box is displayed with the components arranged as they would actually appear on your form.

When you click in the previewed GUI form, mouse events are delivered to the actual components and you can see the components "in action." Thus, for example, you can move sliders, type into text fields, and buttons look "pressed" when you click them, however, cross-component and event handling code is not executed.

How to Create Accessible Forms

Modify the accessibility properties of the components in GUI forms to make them accessible to technologies such as screen readers. Use the procedure to edit a form or the accessibility properties of a component.

To ensure that your GUI forms and the components contained within them meet accessibility requirements, you can adjust their accessibility properties. A GUI is considered accessible when it works with various assistive technologies, such as screen readers.

The following properties can be edited to aid accessibility:

- Accessible Name - Sets the name for the component. By default, the name is set to the component's text property value.
- Accessible Description - Sets the description for the component.
- Accessible Parent - Sets the name of the accessible parent component.

To edit a form or component's accessibility properties:

1. In the Structure window, select the form or component whose accessibility properties you want to modify.
2. In the Properties window, scroll down to the Accessibility properties.
3. Click the ellipsis (...) button to open the Property Editor and then enter the desired value.

   Alternately, you can click the current property value to select it and enter a new value.

Working with Event Handling

Modify the accessibility properties of the components in GUI forms to make them accessible to technologies such as screen readers. Use the procedure to edit a form or accessibility properties of a component.

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 `initComponents()` 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 text property of a button, using the Properties window, 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 Structure window 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.

How to Attach Event Handling Code to Menu Events

Use UI design tools in JDeveloper to attach event handler code to component and menu events. The build of a Java program consists of initialization code that is executed when the UI components are created, and event-handling code that is executed when the user performs an action.

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 a JFrame form in the visual editor.
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 Properties window, expand the **Events** node 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 the menu command.

How to Attach Event-Handling Code to a Component Event

In Swing, a menu item has `actionPerformed` events that are executed whenever the user selects the menu item, and `CheckboxMenuItems` have `itemStateChanged` events. Use the procedure to add code to the event of a menu item.

Using the Events category in the Properties window, 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 GUI builder or in the Structure window.
2. In the Properties window, 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.

How to Quickly Create an Event Handler for a Component's Default Event

An event handler can be created in the GUI builder. Use the procedure to create an event handler for a component default event.

You can create an event handler in the GUI builder.

To quickly create an event handler for a component's default event:

1. Select a component on the Components window and add it to your UI.
2. Double-click the component in the GUI builder. 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.

How to Modify GUI Source Code

The IDE generates blocks of guarded code as the GUI form is created in the GUI Builder. Use the procedure to modify the guarded block of a form component.

The IDE automatically generates grey guarded blocks of code as you create your GUI form in the GUI Builder. However, you can modify the way initialization code is generated and even write custom code to be placed within the initialization code.

You can modify the way initialization code is generated for a component, form, or component property by editing its Code properties in the Properties window. In addition, you can write custom code and specify where it should be placed within the initialization code.

To modify a form component's guarded block:

1. In the Structure window, select the component whose initialization code you want to edit.
2. Scroll down to Code Generation group of properties that lists individual properties for adding custom code to be generated for given component in the guarded code.

   Alternatively, right click the component itself and from context menu choose **Customize Code** which opens a dialog where the custom code of all types can be edited in a more comfortable way.

> **Note:**
>
> You can freely edit all the other code in the Java class that is not guarded, and so customize the UI to your needs. All components have fields, so can be further modified from anywhere. The best place is in the constructor just after the `initComponents()` call. In most cases it is sufficient to write the code here, and it's much easier than trying to get the piece of the code appear inside `initComponents()` by entering it in the correct code property in the GUI builder. That should be left only for really special cases.

Modifying GUI Form Code Outside of the IDE

A form comprises two files, a `.java` file with the Java source code of the form, and a `.form` file that is used to generate the `.java` file when the form changes in the GUI Builder. You can edit the `.java` files in external editors.

In the IDE each form is comprised of two files:

- A `.java` file, which contains the form's Java source code.
- A `.form` file, which stores the information that is used to generate the .java file when you make changes to the form in the GUI Builder. This file does not need to be distributed with your application. If you delete this file, you can no longer use the GUI Builder to change the form.

You can edit the `.java` files using external editors (not while the form is being edited in the IDE), with the following exceptions:

- Do not modify the content of the `initComponents()` method. The body of this method is always regenerated when the form is opened in the IDE.
- Do not remove or modify any of the special comments placed in the source by the IDE's GUI Builder (`// GEN-...`). They are required for the form to open correctly. These comments are not visible inside the IDE's Source Editor.
- Do not modify the headers or footers of event handlers.

How to Modify Code Generation for a Property

Set a property value as custom code. Use the procedure to modify code generation.

A property value can be set as custom code.

To modify code generation:

1. Select the component in the GUI Builder or Structure window to display its properties in the Properties window. Note that if you select multiple components, their common properties are displayed and any modifications apply to all of the selected components.
2. Edit the component's properties in the Properties window by selecting the property and entering the desired value.
3. If the property you want to edit has an ellipsis (...) button, you can click it to open a special property editor that provides more advanced editing options (e.g. a color chooser for a color-type property), or alternate ways of specifying the property value (for example, using a resource bundle for a text property), including a possibility to type directly the code that should represent the property value.

Use this to edit a given property in another way than in the small in-place editor in the properties window. You can also choose from several ways how to enter the property value via the combo box at the top.

Working with the UI Debugger

Besides standard Java and PL/SQL debugger features, JDeveloper provides support for debugging graphical user interfaces for AWT and Swing-based client applications and applets. Use Check for Updates in Help to download, and use 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.

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.

Working with UI Debugger Windows

Of the three dockable windows that JDeveloper provides, the UI Tree and the UI Outline windows open when the UI Debugger starts, and the Events window appears the first time an event is tracked. Select UI Debugger in View to toggle all three 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.

How to Start the UI Debugger

When you start the UI Debugger to perform a task, JDeveloper displays a hierarchical structure of the application in the UI Tree, and a graphical representation of the application user interface in the UI Outline window. Use the procedure to start the UI debugger.

Before performing any UI Debugger task, you must first start the UI Debugger.

To start the UI debugger:

1. Select the project in the Applications window that you want to debug.
2. Select a run configuration.
3. Choose **Run** > **UI Debug** <***projectname***>.**jp**r 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.

Examining the Application Component Hierarchy

The information in the UI Tree and the UI Outline windows are synchronized. A component that is selected in the UI Tree hierarchy is highlighted in the UI Outline window, and a component that is selected in the UI Outline window is displayed in the UI Tree 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.

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.
- 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.
- Note that the components that are not selected in the UI Outline window are shaded red.
- Note that hidden components are represented by gray text in the UI Tree.
- Right-click a component in either windows to display the context menu options. See UI Tree or UI Outline for more information.

How to Display Component Information in the Watches Window

A watch monitors the changing values of variables or expressions as a program runs. Use the procedure 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.

How to Inspect a UI Component in an Properties window

You can view the state of a UI component in a JDeveloper Properties window.

To display a UI component in an Properties 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.

How to Trace Events Generated by Components

Monitor the firing of selected events that are generated by UI components using the event tracing feature. Use the procedure to trace events that are generated by components, and to derive the information to determine the content of events and their sequence.

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.

How to Show Event Listeners

The show listeners feature discovers the recipients of the events that are fired by UI components, and the information can be used to determine the extent of UI events. Use the procedure to trace events that are generated by components.

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.

Remote Debugging GUI Applications

When the JDeveloper debugger that is attached to the program to be debugged is launched, remote debugging proceeds like local debugging. JDeveloper supports the remote debugging of GUI applications by means of the CLI.

JDeveloper supports remote debugging of 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

How to Remote Debug GUI Applications

Remote debugging on a GUI application can be initiated from the CLI. Use the procedure to debug GUI applications remotely.

You can remote debugging a GUI application by entering commands on the command line.

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.

How to start the JDeveloper IDE for Remote UI Debugging

You can remotely debug a project UI in JDeveloper. Use the procedure to start the JDeveloper IDE for remote UI debugging.

You can use the JDeveloper IDE to remote UI debug a project.

To start the 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 as 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.

Automatic Discovery of Listeners

The events that can be tracked by the UI Debugger is dynamically discovered at runtime. If the events adhere to the JDeveloper guidelines, events that are fired by a listener can be tracked.

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 `Xxx`Event class could be:

```
public class XxxEvent
{
   public int getA(){...}
   public String getB(){...}
   public OtherType getC(){...}
```

14

Working with JavaBeans

This chapter describes Oracle JDeveloper support for JavaBeans technology.
This chapter includes the following sections:

- About Working with JavaBeans
- Using JavaBeans in JDeveloper
- Understanding Standard Event Adapters

About Working with JavaBeans

JavaBeans technology can be used to create reusable software components to build Java applets and Java client applications that communicate with business-tier components directly or indirectly through web-tier components.

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 information on JavaBeans, for example, the basic notion of JavaBeans, creating JavaBeans, and what makes a bean, see `https://docs.oracle.com/en/java/javase/17/docs/api/java.desktop/java/beans/package-summary.html`. The tutorial also contains lessons on writing a simple bean, bean properties, manipulating events and other topics.

Using JavaBeans in JDeveloper

Use the Swing GUI builder to build programs with JavaBeans for GUI objects such as user interface objects, data-aware controls, or system facilities. JavaBeans define properties, methods, and events that are open to manipulation or interaction with other components, and are true objects that exhibit encapsulation, inheritance, and polymorphism.

JavaBeans are the Java building blocks used in the Swing GUI builder 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 Components window and reuse with no (or only minor) modifications

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.awt.Component` 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.

How to Implement an Event-Handling Method

In the GUI builder, an event is the event-handling method that implements the class containing the component. Use the procedure to implement the event-handling method.

In the GUI builder, 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 have a button named `jButton1` in a container called `NewJFrame` and you want something to happen when an end user clicks `jButton1`.

To implement the event-handling method:

1. Select `jButton1` in the `NewJFrame` editor.
2. In the Properties window, expand the Events node. Possible events are listed, and `actionPerformed` is the event generated when a button is pressed.
3. From the field next to `actionPerformed` select the default name of the handler, `jButton1ActionPerformed`.
4. JDeveloper switches to the `NewJFrame` source view and inserts an event-handling method into `NewJFrame` that is called when that event occurs.

   The method is called `jButton1ActionPerformed()` by default.
5. Add code into the method to respond to the button press.

The end user sees all of the potential events from `jButton1` listed on the Events page of the Properties window. 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 Properties window. All the end user must do to use your bean is write the code that fills in the event-handling method.

What Happens When You Create an Event-Handling Method

JDeveloper generates additional code in the `Frame1.java` file to handle aspects of event listening by generating an anonymous inner class for the action adapter, instantiating the class in Frame1, and registering itself as a listener for the button1 event.

Behind the scenes, JDeveloper also generates additional code in the `Frame1.java` file to handle the other aspects of event listening:

- It generates an anonymous inner class for the action adapter that implements the `ActionListener` interface.
- It instantiates the class in `Frame1`.
- 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.

Understanding Standard Event Adapters

In Swing GUI Builder, the default set value of the preferences for the Listener Generation Style property can later be changed. Select an option from the Code Style page of the Project Properties dialog to generate the adapter class.

a description of the Listener Generation Style property. It has its default value in Swing GUI Builder preferences where it is already described in the help. This default value is then used for newly created GUI forms. It can be changed then for each form separately: open GUI form, select its root node in Structure window and then in Code Generation properties set the Listener Generation Style property.

You can control how JDeveloper generates the adapter class by selecting the desired option from the Code Style page of the Project Properties dialog (see How to Set Properties for Individual Projects).

How to Create an Event Set

The event-listener interface describes the events of an event set that defines a type of event, what it communicates, and what is required to generate, and listen to the event. Use the procedure to create an event set.

An event set defines a type of event, what it communicates, and what is required to generate and listen to the event. You can create a set of custom events and create an `EventListener` interface and an `EventObject` class to support those events. The event-listener interface describes the events of the event set.

To create an event set:

1. In the Applications window, select the project you wish the bean to be added to.
2. From the main menu, choose **File** > **New from Gallery**.
3. In the New Gallery, in the **Categories** tree, expand **General** and select **Java**.
4. In the **Items** list, double-click **Event Set**.
5. In the Create Event Set dialog, in the **Name** field, enter the name of the event set.

6. Add, edit, or remove events in the **Notifications** field.
7. Click **OK** to add the new event set classes to your project.

How to Make a Component Capable of Firing Events

Components fire events which are then delivered to the components that are to be notified. Use the procedure 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. The means by which components communicate with each other is called **event passing**. Components fire events. The event is then delivered to the components that are to be notified. The notified components can then perform some action based on the event that took place.

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.

15

Getting Started with Developing Java EE Applications

This chapter provides an overview of the Java EE tools and technologies available for your application development.
This chapter includes the following sections:

- About Developing Java EE Applications.
- Using Web Page Tools.
- Using Enterprise JavaBeans and Java Persistence Components.
- Using Oracle TopLink.
- Understanding Secure Applications.
- Working With Applications That Use XML.
- Working With Applications That Use Web Services.

About Developing Java EE Applications

JDeveloper provides wizards, built in source and visual editors, the Components window and the Properties window, and other features to create, assemble, and reuse your web tier and business components to create and edit your Java EE 6 application components.

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, Components window and Properties window, 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 information on Java EE see the Oracle Technology Network (OTN) Java EE documentation at: http://www.oracle.com/technetwork/java/javaee/overview/index.html.

Java EE and Oracle Application Developer Framework

The Rich client framework of ADF Faces offers a rich library of AJAX-enabled UI components for web applications built with JavaServer Faces.

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, with minimal 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.

See *Understanding Oracle Application Development Framework*.

Using Web Page Tools

Use the wizards in the New Gallery to create EJB projects, entities, Java persistence units, session beans, and message-driven beans.

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 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.

See Developing Applications Using Web Page Tools.

Using Enterprise JavaBeans and Java Persistence Components

Use JDeveloper tools to develop the web tier, or the frontend of Java EE applications. Use JDeveloper wizards to create HTML, JSP and JavaServer Faces or Facelet pages, and related files.

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. See Developing with EJB and JPA Components.

Using Oracle TopLink

Oracle TopLink is an object-persistence and object-transformation framework. The framework provides development tools and run-time capabilities to reduce development and maintenance efforts, and to increase enterprise application functionality.

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. See Developing Persistence in Applications Using Oracle .

Understanding Secure Applications

Java EE applications are secured using container-managed security alone. Oracle ADF Security framework provides authentication and authorization services to Fusion web 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. See Developing Secure Applications.

Working With Applications That Use XML

JDeveloper provides an XML source editor, an XML validator, and tools for working with XML schemas to work with the XML files in an application. You can also use JDeveloper to create and edit XSQL files.

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 Components window. See Developing Applications Using XML.

Working With Applications That Use Web Services

Web services constitute a set of messaging protocols and programming standards that expose business functions over the internet using open standards. JDeveloper provides tools to manage existing web services, and develop and deploy new web services.

Web services are 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 accessed programmatically to return a response. JDeveloper provides tools to manage existing web services, and develop and deploy new 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 wizards create the deployment files for the application servers. See How to Create JAX-WS Web Services (Bottom-up).

JDeveloper also supports a set of standard Java-to-XML type mappings. You can also create custom serializers for unique object types. See Using to Create and Use Web Services.

16

Developing Applications Using Web Page Tools

This chapter describes how to use the Oracle JDeveloper tools and features such as page building wizards, visual and source editors, Components window, and Properties window to build and edit your user interfaces and business services using HTML, JSP, and JSF/facelets, expression language, scripting, and servlets.
This chapter includes the following sections:

- About Developing Applications Using Web Page Tools
- Developing Applications with JavaServer Faces
- Developing Applications with HTML Pages
- Working with Java Server Pages
- Developing Applications with Java Servlets
- Developing Applications with Script Languages
- Working with JSP and Facelet Tag Libraries

About Developing Applications Using Web Page Tools

JDeveloper provides source editors, visual editors, integrated component and property tools to add and edit pages, elements and related properties in pages including business service and localization components.

Oracle JDeveloper provides you with a wide range of tools to develop the frontend or view layer of your Java EE applications.

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.

In addition, 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.

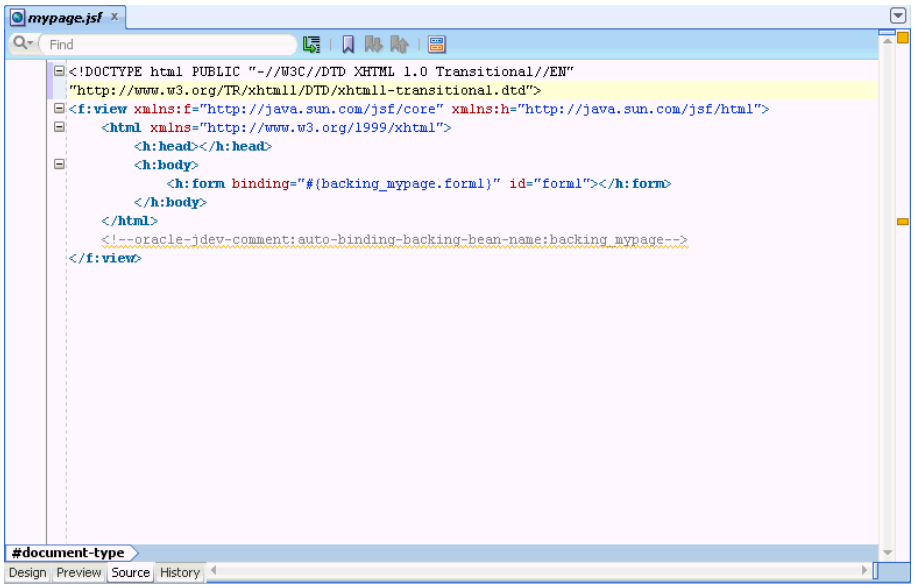
This chapter walks you through the web page tools and step-by-step instructions for many of the tasks you will use to develop your application web pages.

Using the Source Editor

JDeveloper provides the source editor to create custom code for pages, and the visual editor which represents a page visually.

The source editor is your basic code editor for your web pages. Use the source editor to custom code your pages alongside the visual editor which shows a visual representation of your page. The figure displays the source editor for a JSF page.

Figure 16-1 Source Editor with Typical JSF Code



Source Editor Features

The source editor provides features such as Quick Doc for Tags, Code Templates, Code Insight, Jump to Managed Bean, Editor Splitting, and Toggle Line Comments.

The source editor comes with several features to simplify your coding tasks. The following table lists the primary source editor features.

Table 16-1 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.
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.

Table 16-1 (Cont.) Primary Source Editor 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.
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.

Working in the Visual Editing Environment

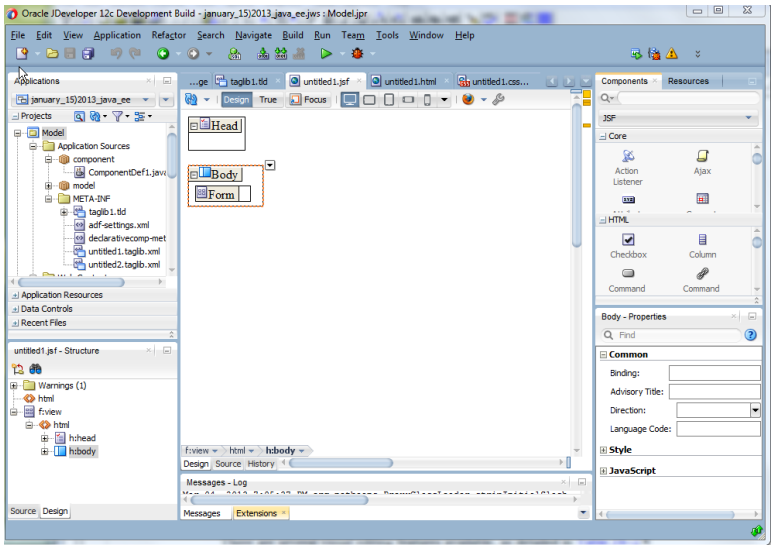
Use the JDeveloper JSP/HTML Visual Editor to edit web pages including JSP, JSF, facelets, and HTML pages.

The JSP/HTML Visual Editor is used for WYSIWYG editing of your web pages including JSP, JSF, facelets, and HTML pages.

Your web page elements are visually displayed or structurally represented in the visual editor. JSP tags including JSTL, and BC4J tags are shown as icon and tag names, while HTML pages are 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 opens up with the Components window alongside it to drop and drag to the page, as shown in the figure.

Figure 16-2 Visual Editor For a JSF Page



Primary Visual Editing Features

JDeveloper supports visual editing features such as Visual and Non-Visual Element Support, Stylesheet Associations, Table and Form Design, and Editor Splitting.

The following table describes available visual editing features.

Table 16-2 Primary Visual Editor Features

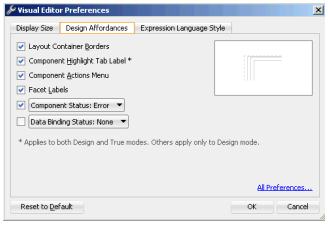
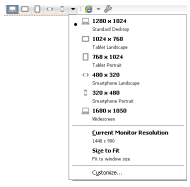
Features	Description
Instant Look and Feel	Opening a file in the visual editor renders it in HTML and associated web formats. You immediately see the results of your edits.
Visual and Non-Visual Element Support	Add visual components and tags to your page, as well as scripting languages, expressions, and values.
Stylesheet Associations	Add or remove stylesheets to your pages, and modify text styles.
Move or Resize Page Elements	Manually drag to resize elements or change page element properties and in the Properties window or by right-clicking and choosing to Properties.
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.
Table and Form Design	Drag Table or Form Components onto the page and visually edit their design.
Modify Element Attributes	Select your page elements and go to the Properties window to add or edit attributes.
Nested Components Appear in Chronological Order	View and select your nested components in chronological order using the associated web formats, just like a browser. Immediately see your results rendered.
Structure Review	View the structure of page elements in the Structure window.
Context Sensitive Editing Tools	Components window tag library shows components available in context only.
Context Command Menu	Right-click anywhere in the page to bring up a menu of commands available for that selection.
Total Tool Integration	The Visual Editor is integrated with the Java Source Editor, the Structure window, Components window, Properties window and Data Binding palette to support the assembly of databound web pages with simple drag and drop operations. Changes to any tools is reflected across the all the tools immediately.

Additional Editing Tools and Features

JDeveloper provides visual editor window tools and features such as Refresh, Design Mode, Preferences, and Browser Selection.

There are also some additional editing tools and features that are available as icons on the top of the visual editor window.

Table 16-3 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.
	Design Mode	Lets you choose whether or not to see design affordances such as facet tables or extra container spacing. True hides affordances and shows the page as it appears at runtime.
	Preferences	Brings up the Visual Editor Preferences dialog. These options are also available by going to Tools > Preferences > JSP and HTML Visual Editor 
	Display Customization	Lets you choose from a variety of custom display sizes for the visual editing window, including a display to match your monitor resolution and size to fit the window.
	Browser Selection	Lets you choose which of your available browsers you prefer to view to use to view your page.

Keystroke Commands

Some JDeveloper features, Breadcrumbs, Component Selection, Editing Containers, Style sheet Linking to HTML files, and Mobile Device Display are accessible by means of simple keystroke commands.

The following table lists the features that are available with simple keystroke commands while you are editing your web pages.

Table 16-4 Primary Visual Editor Keystroke Command Features

Features	Description
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 **Focus 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**.
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 Properties window.
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 Applications window opens or brings the default editor to the foreground. To locate the node in the Applications window that corresponds to the file you are currently working on, right-click and choose **Select in Applications Window**. 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. This can also be used for JavaScript.
Style sheet Linking to HTML files	Link a style sheet to your HTML files simply by dropping a `<style>` or `<link>` tag from the Components window 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.

How to Expand and Collapse Container Elements

Select to show more or less detail on your web page by expanding or collapsing nested JSP and HTML page elements in the visual editor and in the Structure window.

Choose to show more or less detail on your web page by expanding or collapsing nested JSP and HTML page elements in the visual editor and Structure window, as shown in the figures below.

Figure 16-3 Container Tags in Nested Rectangles

Figure 16-4 Collapsed HTML Table

To collapse the container element, choose from one of the following options:

- Click the + (plus) sign of the container element.
- Right-click the container element and choose **Expand All Below** from the context menu.

How to Change the Default Environment Settings

Change the appearance of the display features in the visual editing window using the Preferences page. Use the procedure to change the default settings.

Use the Preferences page to change the look and feel of your visual editing window. The following features are available:

- 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 default settings:

1. From the main menu, choose **Tools** > **Preferences.**
2. Select **JSP and HTML Visual Editor**.
3. Select your options and set fields.
4. Click **OK**.

How to Display Invisible Elements

Hide or show elements that are not shown by default. Use the procedure to change the display of invisible elements.

Choose to hide or not hide elements that are not shown by default, such as:

- HTML named anchors, script tags, and line breaks.
- JSP tag library directives and bean tags.

To change the display of invisible elements:

1. From the main menu, choose **Tools** > **Preferences**.
2. In the Preferences dialog, expand **JSP and HTML Visual Editor** and select **Invisible Elements**.
3. When finished click **OK.** You can toggle the display of invisible elements on and off when you are working. Go the main menu and choose **Design** > **Show** and select **Invisible HTML Elements** or **Invisible JSP Elements**.

How to Execute JSP Tags

Configure tag libraries to execute in the visual editor using the Manage Libraries page. Use the procedure to set a JSP tag library to execute at design time.

To get a snapshot of what the page looks like at runtime, run the tag library in a simulated JSP/Servlet container. Use the Manage Libraries page to configure your tag libraries to execute in the visual editor.

To set a JSP tag library to execute at design time:

1. From the main menu, choose **Tools** > **Manage Libraries**, or select a project and double-click.
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**.

How to Display JSP Tags by Name Only

Display JSP tags by name to get a simpler view of a page, and use the Preference page to hide embedded EL syntax. Use the procedure to display JSP tags by name only.

If you want to get a simpler view of your page, display JSP tags by name only. Go to the Preference page and choose to not show embedded EL syntax. For example, `<c:out value="${Row.Deptno}"></c:out>` displays as `out` vs. `${Row.Deptno}`.

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.

How to Change Keyboard Preferences

Customize the default keymap settings, and specify shortcut key assignments using the Preferences page. Use the procedure to customize keymap shortcut keys.

Use the Preferences page to customize the default keymap settings, and specify the shortcut keys assignments.

To customize keymap shortcut keys:

1. From the main menu, choose **Tools** > **Preferences.**

2. In the Preferences dialog, select **Shortcut Keys**.
3. Click **More Actions** and choose **Load Keyboard Scheme**.
4. Select one of the preset keymaps and click **OK**.
5. The shortcuts for that keymap are listed. Make the change you want, and click **OK.**

How to Select an Element

When you select a single element, a container element, or multiple elements, the selected element is highlighted in the Structure window. Use the procedure to select an element.

Select a single element, or 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 elements, all its contained elements are selected. If you copy, move, or delete the container, all contained elements are included.

   OR

   Right-click the element. When you right-click elements, a context menu of options displays. For example, to highlight the element code in the page source, right-click and select **Go to Source.**

> **Note:**
>
> Double-clicking an element brings up an editor dialog for the tag.

How to Select Multiple Elements

Use the control key to select and manage grouped multiple elements, or deselect an element without losing other selections. Use to procedure to select multiple elements.

Use the control key to select and manage grouped multiple elements. You can also deselect an element without losing other selections

To select multiple elements:

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.

How to Select a Range of Elements

You can select a large range of elements in two clicks. Use the procedure to select a range of adjacent elements.

Use **shift-click** to select a large range of elements in two clicks instead of scrolling through and selecting each element.

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 displays, to easily know which item you want to select.
>
> Press Ctr+Shift+Up to select the container element for an element contained inside another element. This also works to scroll through nested containers. For example, when you add a link to text you will need to press Ctr+Shift+Up twice to move to the link target.

How to Select Text

Use one of the options to select and edit words, strings, or groups of text on the page. Use the mouse pointer for the selection.

Use your mouse to select and edit words, strings, or groups of text on the page.

To select text:

Choose one of the following options:

- Double-click a single word.
- Triple-click an entire line.
- 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.

How to Select Insertion Points in the Visual Editor

A page displays visual cues to locate the insertion point before, after, or contained within a target element. Use the procedure to select an insertion point in the visual editor.

There are visual cues on the page to locate the insertion point before, after, or contained inside a target element.

To select an insertion point in the visual editor:

- When dragging an element, a vertical line | appears to pinpoint your desired insertion point. Release the mouse button to insert that element in that spot.

- Visual clues for insertions:
  - 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.

How to Select Insertion Points in the Structure Window

Move elements around pages using the Structure window, and see elements in chronological order as they are added to the page. Changing their position in the Structure window changes the order on the page.

Move your elements around your pages using the Structure window. Elements are shown in chronological order, i.e., as they were added to the page. Changing their position in the Structure window changes that order on the page.

To select an insertion point in the Structure window, choose from these options to drag a page element to an insertion point:

- 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:**
>
> Disallowed insertion points are indicated by a slashed circle.

How to Insert Elements from the Components Window

The Components window can be used to add UI and data elements to web pages. Use the procedure to insert a page element.

Use the Components window to add UI and data elements to your web pages. When you select an insertion point, the point is selected in the Structure window as well as the page, to help you verify the insertion position visually and hierarchically. See Using the Components Window.

To insert a page element:

1. Select the Components window package, or page from the dropdown list. The Components window is context sensitive and displays only those options that are relevant to the active file.
2. Choose from one of the following options:
   - Select the insertion point and click the element to insert.
   - Drag the element from the Components window to the insertion point.

3. Depending on the element, a corresponding insertion dialog appears, prompting you to select or insert a file, or supply tag attributes. The insertion will fail if the component is not valid at the current insertion point.

When you insert a page element, the source code for the element is automatically generated. When you delete an element, code is deleted.

How to Set and Modify Element Properties

Use the Properties option from the View menu to set or modify the property values for any element that is marked green and gets displayed in the Properties window. Use the procedure to set element properties.

The Properties window displays the properties of selected elements. You can set or modify the property values for any element. Set property values are marked with a green square. If the Properties window is not in view choose **View >Properties window** or use the shortcut Ctrl+Shift+I. To undo changes, from the main menu select **Edit** > **Undo action**. Use the **Set to Default** button to reset a property to its original value. See Using the Properties Window.

To set element properties:

1. Select an element.
2. Select the property. A brief description displays at the bottom.

> **Note:**
>
> You can also use the Find box at the top of Properties window to search for the property.

3. Choose from the following options:
   - 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.**
   - Click the ellipsis (...), to display an editor for that property, for example, a color or font selector. Set the values and click **OK**.

   To display an editor for a property double-click the element.

How to Set a Data Source for a Property

Create databound UI components in the Data Control Palette, or set ADF bindings in the visual editor. Use the procedure to databind an element property.

As an alternative to working with the Data Control Palette to create databound UI components, you can set ADF bindings in the visual editor.

Use the Properties window to set or remove data sources for element properties. From a Value Binding dialog 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 specify an ADF binding as a data source you must first create the binding.

To databind an element property:

1. Select an element in the visual editor or Structure window.

2. Select a data source for that property.
3. Click **Bind to Data**. An EL expression displays in the property value field and an ellipsis button (...) shows.
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.

How to Set Properties for Multiple Elements

The Properties window displays the properties of selected elements, and clicking Union in the Properties window toolbar toggles between displaying the properties of the selected elements and displaying only the properties that the selected elements have in common. Set the properties for multiple elements by selecting from the given options.

By default the Properties window displays all the properties of your selected elements. Click **Union** in the Properties window toolbar to toggle between displaying all the properties of the selected elements and displaying only properties that the selected elements have in common (intersection). Values represented in italics are common properties that have different values.

To set properties for multiple elements:

Choose from one of the following options:

- Hold down the Ctrl key and select each of the elements.
- To change the list of properties displayed by the Properties window, click the Union button in the toolbar:
- Select and edit your property. If the value is shown in italics, the selected elements have differing values. Editing the value of a shared property causes all selected elements to have the same value.

How to Cut Page Elements

Run commands to cut web page elements, place them on the clipboard, and to paste them in another location. Use the procedure to cut a page element.

Cut web page elements using typical commands, and place them on the clipboard to paste in another location.

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 cut page element:

1. Select the page element to cut.
2. Press Ctrl+X or 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 the application and not the system clipboard. If you quit JDeveloper without pasting the element, the cut version of the element is lost.

How to Delete Page Elements

Delete web page elements to delete elements and all that they contain from the system. Use the procedure to delete page elements.

Delete web page elements to remove them from the system completely. This does not change the existing clipboard content. If the element selected for deletion contains included elements, deleting the element also deletes all its contained elements.

To delete page elements:

1. Select one or more page element to delete.
2. Press **Delete** or **Backspace**. You can also right-click and select **Delete**, or choose **Edit > Delete** from the main menu.

How to Copy Page Elements

Run commands to copy page elements, or use the drag copy feature. Use the procedure to copy page elements.

Copy page elements using the typical commands, or use the drag copy feature.

In the visual editor you can also:

- Right-click and 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 copy page elements:

1. Select the page element to copy.
2. Press Ctrl+C. You can also right click and select **Copy**, or choose **Edit** > **Copy** from the main menu.

How to Move Page Elements

Drag an element from one location to another. You can also right-click drag an element to a point in the visual editor or Structure window, and then select Move Here from the context menu.

Most elements can be dragged from one location to another. You can also 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.

How to Resize Page Elements

Resize page elements using the Properties window, or using the context menu for the element. Use the procedure to resize page elements.

Resize your page elements using the Properties window or using the context menu for that element.

To resize page elements:

1. Go to the Properties window 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 Properties window.

How to Create and Edit a Data Table

Use the `h.dataTable` tag to display a data table, and insert the tag on a JSF page using the Create Data Table Wizard. Use the procedure to create and edit a data table.

Use the `h.dataTable` tag to display a data table. The Create Data Table Wizard inserts this tag on a JSF page. This wizard also provides basic 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.

To create and edit a data table:

1. Open a JSF page in the visual editor.
2. In the Components window, select JSF from the dropdown menu.
3. Double-click or drag Data Table from the Components window. 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:data`Table component in the Structure window.
   - In the Properties window, 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.

How to Work with Data Table Columns

Use the right-click context menu to access the options to edit, move, insert, or delete data columns.

Use the right-click context menu to access many of the options to edit, move, insert, or delete your data columns.

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.
- 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.**

> **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**.

How to Work with Panel Grids

Use the Create PanelGrid Wizard to create a grid, and use the `JSF panelGrid` tag to display an HTML table. Use the procedure to create and edit a panel grid to which you can add other components.

Use the JSF `panelGrid` tag to display an HTML table. You can add other components inside the panel grid. Use the Create PanelGrid Wizard to create the grid.

Add the panel grid using the Components window. Once created you can change attribute values set in the wizard, by double-clicking on the `h:panelGrid` component in the Structure Pane. The properties editor opens. Change any values as needed.

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.

To reorder the components, drag and drop the columns in the Structure window or in the visual editor.

To delete a grid or a component in a grid, right-click the component and select **Delete.**

To create and edit a panel grid:

1. Open a JSF page in the visual editor.
2. In the Components window, select JSF from the drop-down menu.
3. Select Panel Grid. The Create PanelGrid Wizard opens.
4. Complete the wizard.

How to Paste Markup Code

Copy and paste source code between JSP and HTML files in the same project or different projects. Use the procedure to paste markup code.

You can copy and paste source code between JSP and HTML 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**.

How to View and Edit Web Page Head Content

HTML head content such as style definitions and the browser window title that are concealed elements on web pages can be displayed by using Head Content from the Show Invisible Elements options for Design. Use the procedure to edit an element in the head section of a page.

HTML head content such as style definitions and the browser window title are invisible elements on web pages.

To show head content choose **Design** > **Show Invisible Elements** > **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. Choose from the following options:
   - Click an element in the head section bar to select, and set or modify the element properties in the Properties window.
   - 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.

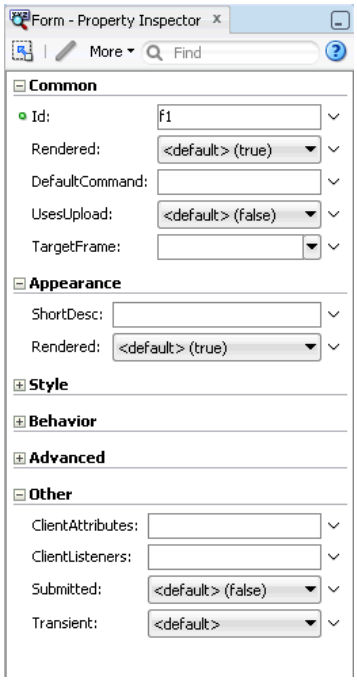
Using the Properties Window

The Properties window displays the properties of a component for editing. The toolbar of the properties window displays the Enable/Disable Auto-Extend tool to toggle on and off the automatic expansion of the window, and Bind to ADF Control to bind or rebind a property to an ADF data control.

Use the Properties window to view and edit the properties of a component.

When you select a component on your editing page the title bar of the Properties window displays the name of the component, as shown in the figure. The main area displays the component properties and their values. If one or more component is selected in the active tool, "Multiple" appears in the title bar, and only the properties shared among the selected components display.

Figure 16-5 Properties Window



The main area of the Properties window displays groups of properties in named sections that you can expand or collapse. Component properties are displayed in fields, dropdown lists, or combo boxes. For boolean properties, a checkbox before the property name 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.

To see a description or more options for a property, right-click a property name, field, dropdown list, or combo box to display a 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.

The Properties window has a few icon tools on the top for common tasks, as shown in Table 16-5.

Table 16-5 Toolbar Icon Features on the Properties window

Icon	Name	Description
	Enable/Disable Auto-Extend	Use to toggle on and off the automatic expansion of the Properties window to display the full contents when the cursor is over the window. When focus moves to another part of the user interface, the window returns to the default position.
	Bind to ADF Control	Click to bind or rebind a property to an ADF data control of your choice.

How to Edit Properties

Edit a property value by entering a new value in a field or by selecting a value from a set of values. When a property value has been edited, the green indicator shows that the default setting has changed.

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. The following are additional options available to edit properties:

- Click the dropdown arrow 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.
- Click the dropdown arrow 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.

How to Write Custom Property Editors

When you start an application, JDeveloper provides the options to select a role. The Customization Developer role provides the option to edit a property value, and selecting Remove Customization from the property menu removes existing customization.

Write your own property editor to customize the display. If your editor supports tags, the editor displays those tags as a fixed set of values. If the editor does not support tags, the system checks to see if custom editing is supported. If neither are supported, a text editor appears.

When you start up the JDeveloper application you are provided the option of selecting your role. Choosing Customization Developer gives you these options:

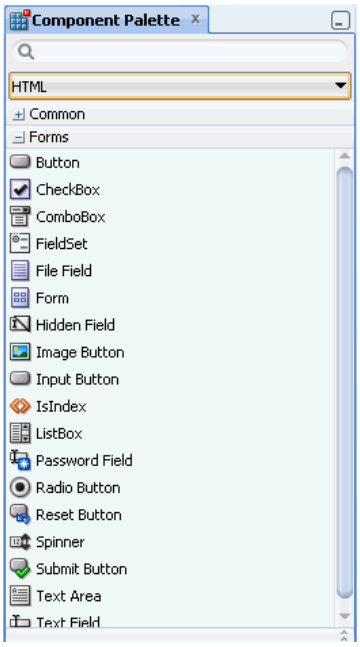
- When you edit a property value, an orange dot appears next to the property name. (Property values that were modified in Default role have green dots next to the properties.)
- From the property menu next to a text-only property, choose **Remove Customization** to remove existing customization that was previously applied in the same customization layer context.

Using the Components Window

The Components window displays component libraries, and provides drag and drop operations. The options that are available depend on the type of file that is open.

The Components window displays your component libraries, and provides simple drag and drop operations. The available components vary depending on the type of file open. For example, if you are editing an HTML file, the Components window displays a list of common HTML components, as shown in the figure.

Figure 16-6 HTML Components in Components window



Using the Components Window

The Components window organizes file and page components as pages. From the options available, Icon View or List View displays components in an icon only view, Add to Favorites adds components to the Favorites panel, and Edit Tag Libraries changes the list of JSP tag libraries.

Your file and page components are organized in the Components window as pages. Select the page you want from the dropdown list at the top.

To insert a component into a file, drag it from the to an insertion point in the editor. In some cases you click a component and click in the editor to insert.

There are a variety of features available:

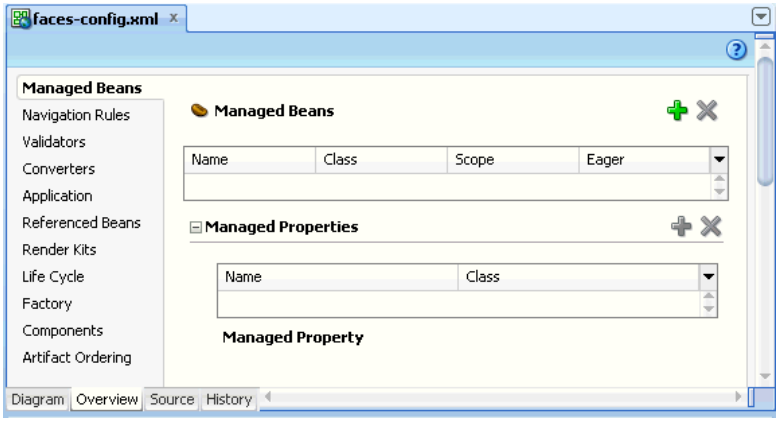
- To search for a component by name, enter the name or part of the name in the binocular icon field and click the green go arrow.
- By default components are displayed in a list view (icon plus name). Change the display to an icon only view by right-clicking a component and choosing **Icon View** or **List View**.
- To add frequently used components, right-click a component and choose **Add to Favorites**. The component is added to the Favorites panel.
- For projects with JSP tag libraries, change the list of JSP tag libraries available for selection by right-clicking in the Components window and choosing **Edit Tag Libraries**.

Using the Overview Editor for JSF Configuration Files

Use the overview editor to add and edit configuration data stored in the `facesconfig. xml` file that displays the main JSF configuration elements, and child elements. Use the procedure to access JSF configuration elements.

Use the overview editor to add and edit configuration data stored in `faces-config.xml`, as shown in the figure.

Figure 16-7 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.

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 in for element list on the left.
- The bottom area of the main panel shows child elements at the top area.

To access JSF configuration elements:

1. In the Applications window, open the workspace that contains your JSF application.
2. In the workspace, open your project.
3. Open the WEB-INF node.
4. 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.

How to Add, Delete, or Edit JSF Configuration Elements

Run commands to see the dialogs that allow you to manage selected elements. Use the New or Delete options to add or delete JSF configuration elements.

Most elements can be managed using simple commands and dialogs that display when the element is selected.

To Add or Delete 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 is an existing class. Select the class by clicking **Browse...** . If no browse button appears, or if the entry is not a class name, 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.

How to Work with JSF Configuration Child Elements

Run commands to see the dialogs that allow you to manage selected elements. You can view, add, delete, or edit child elements.

Most child elements can be managed using simple commands and dialogs that display once the element is selected.

To view, add, delete, or edit child elements:

- **To view child elements.** Select an element from the element list on the left. The main panel displays. Select a 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, the child element must be an existing class. 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, enter a value directly.
- **To edit an existing child element.** Select the element from the table. The properties panel for the element opens to change the value. 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 using right-click delete.

Planning Your Page Flows With JSF Navigation Diagrams

The JSF Navigation Diagrammer can be used to diagram JSF pages and the navigation between the pages.

Use the JSF Navigation Diagrammer to diagram your JSF pages, and the navigation between the pages. When you create a JSF page, the diagrammer is automatically enabled and synchronized with anything you do in the editing tools.

How to View Your Navigation Diagrams

When a navigation diagram is first viewed, a diagram file which includes the JSF configuration file with all the settings is created, and the diagram file and the configuration is included in the version control or source control tool. Use the procedure to view the navigation diagram.

When you first view the navigation diagram, a diagram file is created for diagram details including the JSF configuration file that holds all the settings. If you are using versioning or source control, the diagram file is included as well as the configuration file it represents.

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 Components window also displays. The JSF Diagram Objects page of the Components window shows entries for the elements that can be included on a JSF navigation diagram. To add JSF diagram elements use the Components window.

To view the navigation diagram:

1. In the Applications window, 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 Applications window 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 including the configuration file using the full file name, and the diagram file with the.jsf_diagram extension.

When you first open the JSF configuration file, the configuration file node displayed in the Applications window indicates that there have been changes, even though no changes have yet been made. This is because the node displayed in the Applications window represents both the JSF configuration file and the navigation diagram. 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 Applications window indicates that changes have been made.

If you have a large or complex application, the JSF and related diagram files can be large and loading can take a long time. Choose not to use the diagram as the default editor and no related diagram file is created with your JSFs, unless you specifically request one.

How to Add an Element to a JSF Navigation Diagram

The JSF navigation diagram and the Components window can be used to create a diagram representing the pages in an application, and the navigation cases between them. Use the procedure to add a navigation diagram element.

Use the JSF navigation diagram and the Components window 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 a navigation diagram element

1. Open the diagram.
2. In the Components window > **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.

Figure 16-8 Icon Showing Page is 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. When you have created the page, the icon on the diagram changes to indicate that the physical page has been created.

Figure 16-9 Icon Showing Page is Created

6. Save your changes.

How to Add a JSF Navigation Case to a JSF Navigation Diagram

Add navigation cases to a diagram using the Components window. Use the procedure to add JSF navigation to a diagram.

Use the Components window to add navigation cases.

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.

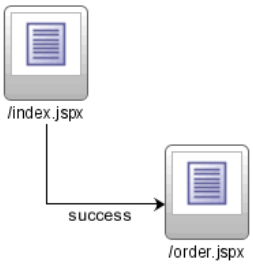
To add a JSF navigation to a diagram:

1. Open the diagram.
2. 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 Components window > 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 the figure.

Figure 16-10 Navigation Case with Straight Line

To draw the navigation case as a line with angles, select either Polyline or Orthogonal in the editor toolbar as shown in the figure.

Figure 16-11 Navigation Case with Angled Lines



5. Save the changes to your JSF navigation diagram and save the changes to the JSF configuration file.

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.

How to Add a Note to a Navigation Diagram

Add notes to a navigation diagram using the diagram annotations feature in the Components window. Use the procedure to add a note to a JSF navigation diagram.

Use the diagram annotations feature in the Components window to add notes to your navigation diagram.

To add a note to a JSF navigation diagram

1. View the JSF navigation diagram for your project.
2. In the Components window, 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

How to Attach Notes to Elements in a Navigation Diagram

Add notes to elements in a navigation diagram using the diagram annotations feature in the Components window. Use the procedure to attach a note to an element in a JSF navigation diagram.

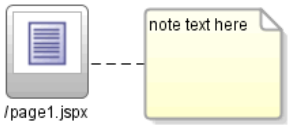
Use the Annotations feature in the Components window to attach notes to your navigation diagrams.

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 Components window, 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 the figure.

Figure 16-12 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.

How to Set Layout Default Styles on a Navigation Diagram

Use the icons on the editor toolbar to define a layout style. Use the procedure to set layout default styles on a navigation diagram.

Use the icons on the editor toolbar to choose your layout style.

To set layout default styles on a navigation diagram

1. Choose a layout style from the editor toolbar. Table 16-6 shows the list of layout styles.
2. Save your changes.

Table 16-6 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.

How to Refresh Your Navigation Diagram to Reflect Changes in the Configuration File.

Refresh a navigation diagram to save changes that were made, and to view the changes. Use the steps to refresh a navigation diagram.

Refreshing your navigation diagram will show changes you have across the editing tools.

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.

How to Use the Navigation Diagrammer to Manage JSF Pages

Use the navigation diagrammer to add, edit, rename, and delete JSF pages. Tabulated are the actions that can be performed on the Diagrammer Page and their effects.

You can use the navigation diagrammer to add, edit, rename, and delete JSF pages. Table Table 16-7 shows the effect of various actions you perform in the diagrammer.

Table 16-7 Diagrammer Page Management Actions and Effects

Action	Effect
Delete	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 Applications window.
Edit	When you edit web pages manually, the JSF navigation diagram and/or the JSF configuration file is not automatically updated.
Rename	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 Applications window, 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.

Editing and Deleting Navigation Cases

The associated configuration files are updated when navigation cases are edited or deleted.

When you edit or delete navigation cases, the associated configuration files are also updated.

Table 16-8 Diagrammer Page Management Actions and Effects

Action	Effect
Delete	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 Navigator.

Table 16-8 (Cont.) Diagrammer Page Management Actions and Effects

Action	Effect
Edit	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. To change the destination for an existing navigation case, delete the existing case and create a new one to the correct destination. If your diagram file is large and takes a long time to open in the diagrammer, you have the option to open the JSF configuration file in another editor.

How to View Navigation Case Properties

View and edit navigation case properties in the Properties window. Use the procedure to view properties of a navigation case.

Use the Properties window to view and edit you navigation case properties.

The navigation cases are displayed on the diagram as solid lines, with the `<from-outcome>` element value displayed as the label.

To view properties of a navigation case:

1. If the Properties window is not displayed, open it from the **Window** menu.
2. Select the navigation case with properties you want to view. The properties of the navigation case are shown in the Properties window.

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. Use the procedure to publish a diagram as a graphic.

You have the option to save your diagrams as `.jpg`, `.png`, `.svg`, or `.svgz` files to use in documents, or on web pages. Images saved in `.jpg` format 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, then choose **Publish Diagram**.

   Alternatively, click on the surface of the diagram, then choose **Diagram** > **Publish Diagram.**
2. Using the Location drop-down list, and 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**.

How to Use Code Insight For Faster Web Page Coding

Code Insight provides coding assistance by means of completion insight and parameter insight for efficient coding because you can quickly insert code parts, parameters, and elements from

a dynamic list of available options. Use the procedure to use Code Insight on a web page in the source editor.

Use Code Insight to speed up your coding and quickly insert code parts, parameters, and elements from a dynamic list of available options.

Code Insight provides two types of coding assistance: completion insight and parameter insight. Completion insight completes regularly used code snippets for you. Parameter Insight with provide you with a quick-pick list of parameter options available in that instance.

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 (that is, `<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, for example, `<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 CSS files. To use Code Insight for your CSS file, place the cursor inside any <STYLE> tag, then type the open angle-bracket and press Ctrl + Space. A list of possible completions displays. You can filter the available completions by typing the first character of the element.

To use Code Insight in a JavaScript or CSS file:

- For JavaScript files, place the cursor inside any <SCRIPT> tag, then type the open angle-bracket and press Ctrl + Space. For CSS files place the curser inside the <STYLE> tag and type the open angle-bracket and press Ctrl + Space.

JDeveloper displays 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, and so on).

Code Insight will also prompt for completion inside JavaScript-specific XML attributes.

Developing Applications with JavaServer Faces

Within the Java EE platform, JDeveloper supports tools for user interface development using JavaServer Faces technology. Use the procedure to quickstart a JSF application.

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.

To quickstart your JSF application end to end:

1. Build a web application.
2. Create your JSF pages using the New Gallery JSF wizard.
3. Choose a Business Service.
4. Create the backing beans for your business services.
5. Bind the interface components to data.
6. Add application resources and managed beans to `faces-config.xml`.
7. Run your JSF pages.

Building Your JSF Application

Build web pages using the Fusion Web Application, Java EE Application, or Generic Application templates. The selection you make from the technologies offered in the New Gallery Wizards determines the project folders that are created, and the libraries that are added to the folders.

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 using the application templates. Choose from a combination of technologies offered in the New Gallery Wizards. The application you choose determines the project folders created, and the libraries added to the folders as shown in Table 16-9.

Table 16-9 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)

Table 16-9 (Cont.) Web Application Templates

Application	Description
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.

How to Build Your Application Framework

The JDeveloper wizards help you build customized application frameworks. Use the procedure to create a web application and project for a JSF application.

Start by using the wizards to build your customized application framework.

To create a web application and project for a JSF application:

1. From the main menu select **File** > **New** > **From Gallery > General > Applications**.
2. Select an application to create.
3. Complete the steps. The project folders, Model and ViewController, are created and listed in the Applications window under the new application node. If you chose Generic Application, only a Project folder is shown.
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.

How to Create Your JSF Pages and Related Business Services

Use the ADF Faces page templates or quick start layouts to create an application framework, and then use the page building, editing, and modeling tools to create application pages. Use the procedure to create JSF pages.

Once you have created the framework of your application, get your pages up and running with the page building, editing, and modeling tools. 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.

To create your JSF pages:

1. In the Applications window, 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.

How to Choose a Business Services

Use Enterprise JavaBeans, JavaBeans, or Oracle TopLink to map Java classes and EJBs to database tables for business services. Use the procedure to create a business service by

selecting from the Business Service New Gallery options, or to expose the entities in the project.

For your business services you have the option of using Enterprise JavaBeans (EJB), JavaBeans, or 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.

Open the New Gallery and use the provided wizards and dialogs to create your business service, or in the case of web services, to expose the entities in your project, as shown in Table 16-10.

Table 16-10 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.
2. In the Applications window, 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 and follow the steps in the wizard.

Building your JSF Business Component Framework

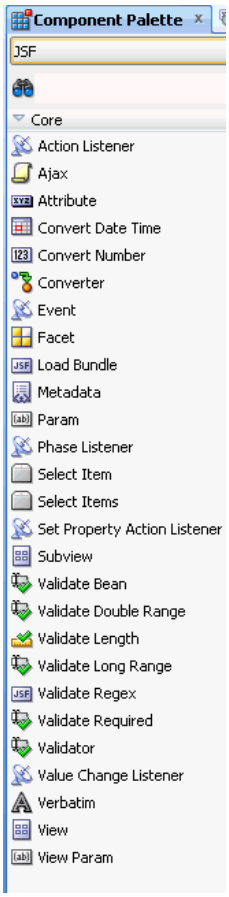
Use the Components window to drag and drop standard JSF components, such as validating and converting components and Message components, onto JSF pages. When a JSF page is created, JDeveloper creates the backing beans and automatically binds them to all the components and corresponding properties on the page.

The Components window contains your standard JSF components to drag and drop onto your JSF pages. When you create a JSF page, the backing beans are created and automatically bound to all of the components and corresponding properties you put on the page.

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, there are validating and converting components that are configurable through the Properties window, as well as a Message component to help you set up the error message output for your JSF pages, as shown in the figure.

Figure 16-13 Core JSF Components Available in Components window



Using the JSF Tag Libraries

Tabulated are the Standard JSF Core Tag Library elements that JDeveloper supports.

The components available on the Components window correspond to the JSF 2.0 facelets tag library. The tag descriptions are detailed in Table 16-11.

For complete information on this and all JavaServer Faces 2.1 component tags and API, see the Oracle Technology Network (OTN) online documentation libraries.

JavaServer Faces 2.1 Facelets Tag Library Documentation: `http://docs.oracle.com/javaee/6/javaserverfaces/2.1/docs/vdldocs/facelets/`

JavaServer Faces Technology 2.1 JSP Tag Library Documentation:

`http://docs.oracle.com/javaee/6/javaserverfaces/2.1/docs/vdldocs/jsp/`

Table 16-11 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 16-11 (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 16-11 (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.
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.

Table 16-11 (Cont.) Standard JSF Core Tag Library Supported Elements

Component Tag	Syntax	Description
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 constraints 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.
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.

Table 16-11 (Cont.) Standard JSF Core Tag Library Supported Elements

Component Tag	Syntax	Description
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.
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.

Using Standard JSF Component Tag Attributes

Set component tag attribute in the Properties window. Add to the JSF page a component and supported attributes for the component tag grouped in the Common, Appearance, Style, and Javascript categories.

Use the Properties window to view and set your component tag attribute. 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 Properties window 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`.

How to Create Managed Beans

Beans that contain logic and properties for UI components on a JSF page are called Managed, or Backing beans. Use the Managed Bean option on the Create JSF page dialog to automatically bind backing beans.

Managed, or Backing beans are beans that contain logic and properties for UI components on a JSF page. Use the Managed Bean tab of the Create JSF page dialog to automatically bind your backing beans, as shown in the figure below. When this option is selected, a default bean is created (or select a managed bean of your choice) for the page you are creating, and then automatically binds all the page components to a corresponding property in that bean. It also creates the associated accessor methods.

Table 16-12 Managed Bean Binding Options

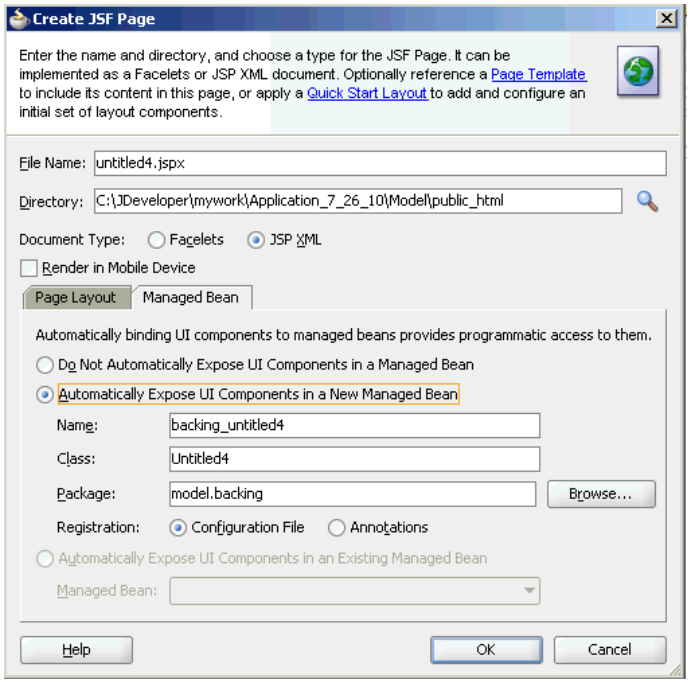
If you want to...	Then choose...
Use a default managed bean for a JSF page	**Automatically Expose UI Components in a New Managed Bean**. Accept the default names, or enter names of your choice.
Use an existing managed bean of your choice for a JSF page	**Automatically Expose UI Components in an Existing Managed Bean**. Then select a managed bean from the dropdown list.

Creating a page and choosing to automatically bind components provides the following features:

- 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.

Figure 16-14 Create JSF Dialog - Create Managed Bean Tab



Creating Managed Beans

Enter method binding expressions, and select existing methods on the page backing bean for component tags with attributes that require method binding using the Properties window. Use the procedure to create managed beans.

For component tags with attributes that require method binding, use the Properties window 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 create managed beans:

1. Create a JSF or JSPX page from the New Gallery.
2. Select the Managed Bean tab.
3. Select **Automatically Expose UI Components in a New Managed Bean**. A new backing managed bean is created with the same name as the JSF page. It is located in the model.backing directory.
4. Add or delete component tags as needed to the JSF page. Edits automatically updated in the backing bean.

How to Create Managed Beans with the Overview Editor

Managed beans for JSPX pages can be created using the XML Overview Editor. Use the procedure to create managed beans with the overview editor.

Create your managed beans for your JSPX pages using the XML Overview Editor.

To create managed beans with the overview editor

1. In the Applications window, double-click to open 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 editing window displays.
3. In the element list on the left, select **Managed Beans**.
4. Click the plus symbol 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**. 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 Applications window.

How to Add Methods to a Managed Bean

Methods for beans can be created using the Events menu. Use the procedure to add methods to a managed bean.

Create your bean methods using the Events dropdown menu.

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.Applications window
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.

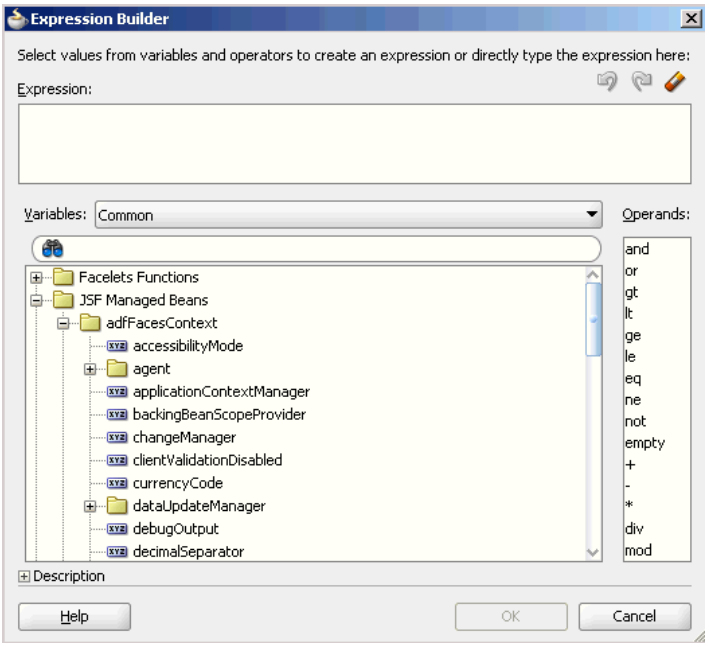
How to Bind Components with EL Expressions

JavaServer Faces provide an expression language for use in JSF pages that provides access to JavaBeans components in beans on a page and in a web application. Add a component to a page, select the component, and create the bindings from the Properties window.

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 Properties window.

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 the figure.

Figure 16-15 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.

Constructing an EL Expression

Either use the component field in the Properties window to edit EL expressions, or use Expression Builder after clicking the component. Use the procedure to construct an EL expression that uses JSF technology.

You can edit your EL expressions from the component field in the Properties window. Click inside the field to see long expressions. Click Ctrl+Space to invoke Code Insight from the Properties window field. You can also add or edit EL expressions by slow-clicking the component and clicking **Expression Builder**.

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 Properties window, select the attribute name.
4. In the attribute action dialog select Expression Builder.
5. Build your expressions and click **OK**.

How to Create Composite Components

A template type that acts as a component and contains a collection of markup tags and other components is a composite component. Use the procedure to create a composite component.

A composite component is type of template, that acts as a component and contains a collection of markup tags and other existing components. It is a reusable component that you create for your application for specific and defined functionality. Your composite component can have validators, converters, and listeners attached to it like any other component.

You can use the New Gallery wizard to create your Composite Components.

To create a composite component:

1. Go to **File** > **New** > **From Gallery**.
2. Expand the Web Tier node and select **JSF/Facelets**.
3. From the Items list, select **Composite Component** and enter the information as directed in the Create JSF Composite Component dialog.

Using Automatic Component Binding

Automatic component binding in a page determines how method binding expressions for the attributes of command and input components such as `action`, `actionListener`, and `validator` are entered.

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 16-15 to choose the component property that will be bound.

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 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 skeleton methods are automatically created.

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.

If you created a JSF page with the file name `myfile.jsp` and you have selected to automatically create a default managed bean, then a backing bean is created as `.backing.Myfile.java`, and placed 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 the coded examples below.

```
...
<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>
...
```

```
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>
...
```

Turning the Automatic Bind Option On and Off

Turn off or turn on the automatic bind option, or change the managed bean selection during binding. Use the procedure to turn off or on automatic component binding.

While editing, 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.

How to Set a Page to Auto Bind to Managed Beans

Use the Design option from the menu to enable component binding so that a new component that is inserted is bound to the selected managed bean. Use the procedure to set automatic binding for a page.

Use the menu bar Design option to enable component binding. All existing bound components and any new components that you insert are bound to the selected managed bean

To set automatic binding for a 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. .

How to Value Bind a Component to a Property

The Properties window can be used to value bind a property. Use the procedure to value bind a component to a property.

Use the Properties window to value bind a property.

To value bind a component to a property:

1. In the visual editor, select the component.
2. In the Properties window, 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.

How to Manually Bind Component Instances to Properties

Bind component instances to properties using the Properties window. Use the procedure to manually bind component instances to properties.

Use the Properties window to bind component instances to properties.

To manually bind component instances to properties

1. In the visual editor, select the component.
2. In the Properties window, 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.

How to Bind an Existing Method with Auto Component Binding

Bind existing methods with auto binding by using the Properties window. Use the procedure to bind to a method with auto component binding on.

Use the Properties window to bind existing methods with auto binding.

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 Properties window, 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.

How to Bind a New Default Method with Auto Binding On

Bind new default methods with auto binding off by using the Events menu. Use the procedure to bind to a new default method with auto component binding on.

Use the events dropdown menu to bind new default methods with auto binding off.

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.

How to Bind a New Default Method with Auto Binding Off

Bind new default methods with auto bind off by using the Properties window. Use the procedure to bind a new default method with auto binding off.

Use the Properties window to bind new default methods with auto bind off.

To bind a new default method with auto binding off:

1. In the visual editor, select the component. In the Properties window, 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.

Using Localized Resource Bundles in JSF

Use Default Project Properties dialog to add or remove resource bundles with the content that is built in the JSF application components. You can select the bundles for a project, and text objects in the resource bundle can also be assigned a key value string to uniquely identify them.

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.

How to Use Localized Resource Bundles in JSF

Drag the `LoadBundle` component from the Components window to a JSF page to add resource bundles. Use the procedure to use localized resource bundles in JSF.

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}`.

Add your resource bundles to your JSF pages dragging the **LoadBundle** component from the Components window.

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 Applications window, double-click `faces-config.xml` to open it in the JSF Configuration Editor. Switch to the Overview.
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 Components window, select **JSF Core** from the dropdown list, then drag and drop **LoadBundle** to the page. A dialog appears to enter the base name of the resource bundle, and any name for the map variable used in request scope.

   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>
```

How to Work with Facets

When you create complex components, output tags are automatically created and inserted into the facets that the component uses. Use the procedure to add and edit facets.

Many components use facets. When you use wizards to create complex components (such as a table or panel), output tags are 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 add and edit 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 Properties window 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 Properties window to set attribute values.

How to Build JSF Views with Facelets

JDeveloper supports facelets technology that supports features such as faster UI development and deployment, compilation, compile time validation, functional extensibility of components and server-side technologies through customization and code reuse.

Facelets technology is supported by JDeveloper.

The following features are available:

- Reduces UI development and deployment time.
- Faster compilation 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 16-13 Facelets Tag Libraries Included with JDeveloper

Tag Library	URI	prefix	Example	Contains
JSF UI Tag Library	http://docs.oracle.com/cd/E17802_01/j2ee/javaee/javaserverfaces/2.0/docs/pdldocs/facelets/ui/tld-summary.html	ui:	ui:component ui:insert	This tag Library is used for templating
JSF HTML Tag Library	http://docs.oracle.com/cd/E17802_01/j2ee/javaee/javaserverfaces/2.0/docs/pdldocs/facelets/	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.

Table 16-13 (Cont.) Facelets Tag Libraries Included with JDeveloper

Tag Library	URI	prefix	Example	Contains
JSF Core Tag Library	http://docs.oracle.com/cd/E17802_01/j2ee/javaee/javaserverfaces/2.0/docs/pdldocs/facelets/	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://docs.oracle.com/javaee/5/jstl/1.1/docs/tlddocs/	fn:	fn:toUpperCase fn:toLowerCase	JSTL 1.1 Functions Tag Library

Creating a Facelet

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 **OK**.

-
```
<context-param>
<param-name> Facelets.VIEW_MAPPINGS </param-name>
<param-value> *.xhtml</param-value>
</context-param>

<context-param>
<param-name&gt;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.

How to Register a Standard Converter Using a Supplied Tag

A converter component converts and validates JSF input data. Use the procedure to register a JSF standard converter using a supplied tag.

To convert and validate your JSF input data there is a converter component to register a named converter instance, a convert number, and convert date and time in the Components window.

You can configure your converter and validator properties in the Overview editor for your `faces-config.xml` file.

To register a JSF standard converter using a supplied tag:

1. In the visual editor, select the component to register a standard converter.

2. In the Components window, select **JSF Core** from the dropdown list, then click a standard converter. (for example, **convertDateTime**).
3. In the Properties window, set the attributes for the converter.

```
<h:inputText id="hiredate" value="#{employee.hireDate}"
  <f:convertDateTime dateStyle="full"/>
  <f:convertDateTime dateStyle="full"/>
 </h:inputText>
```

How to Register a Standard Converter That Does Not Have Tag

A converter can be registered using either the Components window or the Properties window. Use the procedure to register a JSF standard converter that does not have its own tag.

Use the Components window or Properties window to register your converter.

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 Components window, 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 (for example, j**avax.faces.Integer**). Click **OK**. This inserts the `f:converter` tag in the page. Instead of using the `f:converter` tag, you can use the Properties window to enter the converter ID on the component converter attribute.

```
<h:inputText id="age" ...>
 <f:converter converterId="javax.faces.Integer" />
</h:inputText>

<h:inputText id="age" converter="javax.faces.Integer" />
```

How to Register a Standard Validator Using a Standard Tag

Register a validator with a standard tag using either the Components window or the Properties window. Use the procedure to register a JSF standard validator on a component using a standard tag.

Use the Components window or Properties window to register your validator with a standard tag.

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 Components window, select JSF Core from the dropdown list, then click the standard validator of your choice (for example, **ValidateLength**).
3. In the Properties window, 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.

```
<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>
```

How to Display a Message Next to the Component that Generated the Conversion or Validation Errors

Assign an ID and add the message for a component by using the Properties window and the Components window. Use the procedure to display a message next to the component that generated the conversion or validation error.

Use the Properties window and Components window to assign an ID and add the message.

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 Properties window to assign a unique ID to the component to show a message.
3. In the Components window, 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 Properties window, set the attributes for the message tag.

```
<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>
```

How to Register a Custom Converter or Validator in the JSF Application Configuration

Register custom converters using the Overview Editor. Use the procedure to register a custom converter or validator in the configuration file.

Use the Overview Editor to register your custom converters. 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 register a custom converter or validator in the configuration file:

1. In the Applications window, double-click the application's `faces-config.xml` file to open it in the JSF Overview 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.

How to Edit a Custom Converter or Validator in a Configuration File

Edit a custom converter or validator using the Overview Editor. Use the procedure to edit a custom converter or validator.

Use the Overview Editor to edit your custom converter or validator.

To edit a custom converter or validator:

1. In the Applications window, double-click the application `faces-config.xml` file to open it in the Overview 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.

How to Delete a Custom Converter or Validator in a Configuration File

Edit a custom converter by using the Overview Editor. Use the procedure to delete a custom converter or validator in the JSF application configuration file.

Use the Overview Editor to edit a custom converter.

To delete a custom converter or validator in the JSF application configuration file:

1. In the Applications window, 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.

How to Register a Custom Converter on a Component Using a Converter Identifier

Register a custom converter on a component using the Components window. Use the procedure to register a custom converter on a component using a converter identifier.

Use the Components window to register your custom converter on a component.

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 Components window, 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 Properties window to enter the registered converter ID.

```
<h:inputText id="memberNumber" ... >
 <f:converter converterId="customConverter"/>
</h:inputText>

<h:inputText id="memberNumber" converter="customConverter"/>
```

How to Register a Custom Converter on a Component Using a Value Binding Expression

Register a custom converter on a component with a value binding expression using the Properties window. Use the procedure to register a custom converter on a component using a value binding expression.

Use the Properties window to register your custom converter on a component with a value binding expression.

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 Properties window, 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.

```
<h:inputText id="age" converter="#{someBean.someProperty}" />
```

How to Register a Custom Validator Instance on a Component

Register a custom validator instance on a component using the Components window. Use the procedure to register a custom validator instance on a component.

Use the Components window to register a custom validator instance on a component.

To register a custom validator instance on a component:

1. In the visual editor, select the input component to use.
2. In the Components window, select JSF Core or ADF Faces Core page from the dropdown list, and then click the Validator component.
3. In the Properties window, select a registered validator identifier from the dropdown list, or enter a binding expression. Click **OK**.

```
<h:inputText id="name"
    value="#{MyBean.name}"
    size="10" ... >
  <f:validator validatorId="customValidator" />
  <f:attribute name="someName" value="someValue" />
</h:inputText>
```

How to Bind a Component to a New Validator Method

Bind a component to a new validator method by using the Bind Validator property dialog. Use the procedure to bind a component to a new validator method.

Use the Bind Validator property dialog to bind a component to a new validator method.

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.

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...
}
...
```

Using the Standard Converter and Validator Tags and Syntax

Tabulated are the JSF Standard Converter Tags that JDeveloper supports.

All of the attributes supported by JDeveloper are shown in Table 16-14 and Table 16-15. 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 16-14 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 16-15 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"] />

How to Display Error Messages

Use the Message component to create and define error messages, and use the Properties window to define the attributes. Use the procedure to display one error message next to a component that generated an error.

Create and define error messages using the Message component and the Properties window 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. In the Properties window, assign a unique ID to the component to show a message.
3. In the Components window, 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.
4. Click the column next to **For*** and type the component ID. Then click **OK**.
5. In the Properties window, set the attributes for the message tag.

```
<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 do not have to add individual `h:message` tags manually.

How to Display All Error Messages Generated in a Page

Set the display attributes for messages by using the Properties window. Use the procedure to display all error messages generated on a page.

Use the Properties window to set the display attributes for your messages.

To display all error messages generated in a page:

1. Open your JSF page in the visual editor.
2. In the Components window, 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 Properties window, set the attributes for the **Messages** tag.

```
<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 do not have to add the tag manually.

How to Replace the Standard Message Texts in JSF

Create text resource bundles and then edit the configuration file, and add the bundle classpath using the overview editor. Use the procedure to replace the standard message texts in JSF.

Create your text resource bundles and then use the overview editor to edit your configuration file and add the bundle classpath. JSF first looks for messages in any registered resource bundle before looking into the JSF standard bundle. This lets you 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 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 Applications window, 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. J

How to Add Information About a Form Field to Which a Message Refers

Add a phase listener to a configuration file using the Overview Editor. Use the procedure to add information about a form field to which a message refers.

Use the Overview Editor to add a phase listener to your configuration file.

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 Applications window, 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 Components window, 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.

How to Change the Appearance of Error Messages a JSF Page

Change the appearance of a message using the stylesheet. Use the procedure to change the appearance of error messages on a JSF page.

Use the stylesheet to change you message appearance.

To change the appearance of error messages in a JSF page:

1. Open a JSF page in the visual editor.
2. Link a CSS stylesheet to your page.
3. Select the `h:message` or `hmessages:` component.
4. In the Properties window, 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 Properties window click the column next to `ErrorClass`, then select a style class.

> **Note:**
>
> To use one or more inline styles, expand ErrorStyle in the Properties window; then enter or select a value next to the style you want to specify, for example, background-color.

```
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>
...
```

Configuring JSF Applications

When referenced beans are configured using the `faces-config.xml`, design-time tools work without the beans that are unavailable at design time but become available at runtime. When a JSF or facelet file is created, a `WEB-INF/faces-config.xml` is automatically created.

Configure your referenced beans using the `faces-config.xml`. 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.

When you create any JSF or facelet file a `WEB-INF/faces-config.xml` is automatically created. You can have more than one JSF configuration file. You might want to create multiple configuration files for separate areas of your application. Additionally, if you choose to have packaged libraries containing custom components or renderers, you need a separate `faces-config.xml` file for each library. In this case, the configuration file is stored in the META-INF directory (as opposed to the WEB-INF directory).

How to Use the Overview Editor to set the <application> Element

Configure child elements by using the Application section of the Overview Editor. Use the procedure to use the overview editor for configuration files to set the `application` element.

Use the Application section of the Overview Editor to configure 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 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.

How to Add a Bean to a JSF Configuration File

Use the procedure to add a bean to JSF configuration file.

1. In the Applications window, 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.

How to Create a New JSF Configuration File

Create a new JSF configuration file by using the New Gallery wizard. Use the procedure to create a new JSF configuration file.

Use the New Gallery wizard to create a new JSF configuration file. You can then include this configuration file in the `.jar` file that you use to distribute your components or classes.

To create a new JSF configuration file:

1. In the Applications window, select your project. The project contains a WEB-INF node, which contains the file web.xml.
2. Right-click the project node and choose **New**.
3. In the New Gallery, go to **Categories**, expand the **Web Tier**, 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 to purpose your file. If you are adding a configuration file for your application:
    a. Enter a **File Name** for the new configuration file.
    b. Verify or change the **Directory.**
    c. Check the **Add Reference to web.xml** checkbox. When selected, a new `web.xml` file is added, and is read as part of your application configuration.
    d. 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`:
    a. Set the file name to `faces-config.xml`.
    b. Change the **Directory Name** to `META-INF`.
    c. Clear the **Add Reference** to **web.xml** checkbox.
    d. Click **OK**. This creates a new configuration file using the entered name.

How to Edit a JSF Configuration File

Edit a JSF configuration file by using one of the three editors. Use the procedure to edit a JSF Configuration File.

Use one of three editors to edit your JSF configuration file.

To edit a JSF Configuration File:

1. In the Applications window, locate the `faces-config.xml`configuration file located in the `WEB-INF` node.
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:
    - JSF navigation diagrammer, click **Diagram.**
    - Overview editor for JSF configuration files, click **Overview**.
    - XML source editor, click **Source**.

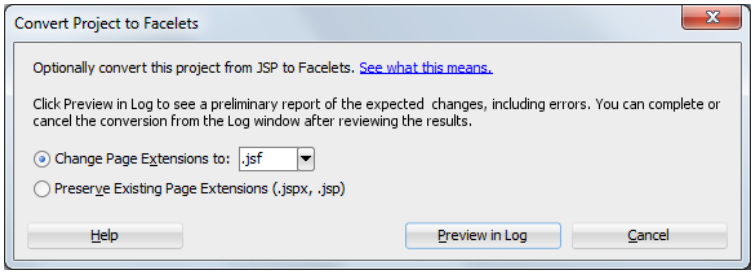
Converting a Project to Facelets

When you convert a project or file to facelets using the Convert to Facelets feature, tag libraries are added, tag usage is changed, and file names are changed.

You can convert a project or file to facelets using the **Convert to Facelets** feature as shown in the figure. This process modifies your project or file in three ways:

- **Tag libraries added** - All facelets tag libraries with the same namespaces as JSP tag libraries used by the Project are added.
- **Tag usage changed** - Each JSP page in this project (`.jspx` or `.jsff`), is replaced with a facelets page.
- **File names changed** - You specify which file extension to use for facelets in this conversion. If the file extension is not currently associated with facelets in this project (javax.faces.FACELETS_VIEW_MAPPINGS context parameter in web.xml) and in the Preferences (**Tools > Preferences > File Types**), then the required mappings are added to `web.xml` and preferences. All .jspx files are renamed to the file extension you specify here. All references to your renamed `.jspx` files are updated accordingly.

Figure 16-16 Convert to Facelets Dialog



Things to Consider Before Converting

Unconvertible JSP tags are left unchanged, JSP pages in XML syntax are supported, conversions cannot be undone, and converting to facelets modifies application preferences. Also to be considered before conversions are that file names are changed, and if JSP file extensions are preserved, the extension functions as facelets after conversion.

The following should be considered before you convert to facelets:

- JSP tags that cannot be converted are left unchanged in the converted page. Until these JSP tags are manually converted, the converted page is not executable.
- The JSP pages in XML syntax, which typically have `.jspx` file extension, are supported. JSP pages in JSP syntax, which typically have `.jsp` file extension, need to be converted to XML syntax in order to use the facelets conversion feature.
- The conversion is not undoable. We recommend that you make a back-up copy of the Project before running Facelets conversion.
- Converting to facelets modifies your application preferences, which applies to any project you use in JDeveloper.
- File names are changed unless you choose to preserve the file extension. Changed file names can cause problems with the version control system in some cases. We recommend that you test converting a single JSP file before converting many files in a project to make sure that the version control system works okay with file name changes.

- If you choose to preserve the JSP file extension (`.jspx`), all files with `.jspx` extension will be recognized as facelets after conversion. If you want to leave some of the `.jspx` files unconverted and run them as JSPs, then manually rename the `.jspx` files to a new file extension for JSP documents, and ensure that the file name change does not cause any problem with the version control system.

Previewing your Conversion Status in the Log Windows

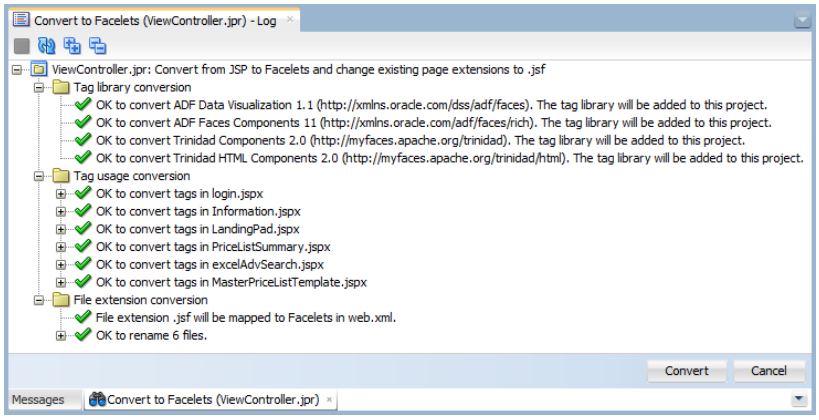
Sections in the log windows present changes being made for the conversion, for the tag libraries conversion, tag usage conversion, and file name conversion.

The changes being made for the conversion are presented in three sections in the log windows, one for the tag libraries conversion, tag usage conversion, and file name conversion, as shown in the figure.

> **Note:**
>
> To the log window, select **Window > Log**.

During your preview session, you can choose to cancel the entire conversion process and back out of any changes being made. To continue the conversion, click **Convert**.

Figure 16-17 Log Window Showing File Names Being Converted



How to Convert your Project to Facelets

A project can be converted to facelets from the ViewController node in the project Applications window. Use the procedure to convert a project to facelets.

Convert your project to facelets from the project Applications window ViewController node.

To convert your project to facelets:

1. In your project Applications window, right-click the ViewController node.
2. Choose **Convert to Facelets**.

Running and Testing JSF Applications

The Integrated WebLogic Server enables you to run and test web applications from JDeveloper.

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.

How to Run and Test Individual Packages

Run a JSF page using the context menu of the page. Use the procedure to run and test individual pages.

From your JSF page use the context menu to run that page.

To run and test individual pages:

1. In the Applications window 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.

How to Run and Test an Entire Project

Specify a default run target and run a project using the context menu, and JDeveloper compiles the application, starts the Integrated WebLogic Server processes, and launches the application. Use the procedure to run and test an entire project.

You can run a project using the context menu.

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.

To run and test an entire project:

1. In the Applications window, 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.

Developing Applications with HTML Pages

JDeveloper supports application development with HTML technology in accordance with the HTML 5 W3C specification.

JDeveloper provides full support for developing applications with HTML technology in accordance with the HTML 5 W3C specification at `http://www.w3.org/TR/html5/`.

There is 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 Using the Source Editor and Working in the Visual Editing Environment.

Building Your HTML Pages

Create a web application and then build HTML pages.

To get started with your HTML web pages, you first need to create a web application. Go to Table 16-9 to see the available application types.

Once you have created your web application framework, you are ready to start building your HTML pages.

How to Create an HTML Page

Create HTML pages by using the New Gallery wizard. Use the procedure to create an HTML page.

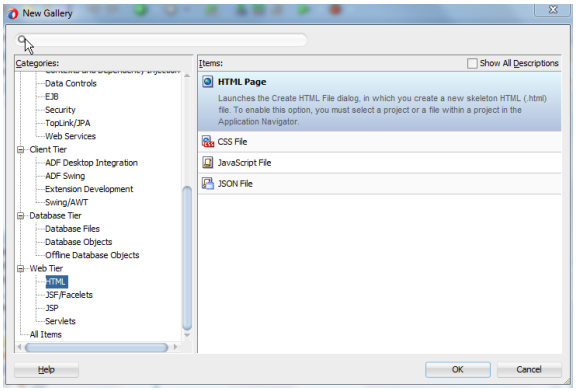
Use the New Gallery wizard to create your HTML pages. A simple HTML file is generated and appears in your active project. The deployment descriptor file `web.xml` is also added to your project. The deployment descriptor file is used by the Integrated WebLogic Server when you run the HTML.

To create an HTML page:

The New Gallery wizard walks you through building your web pages.

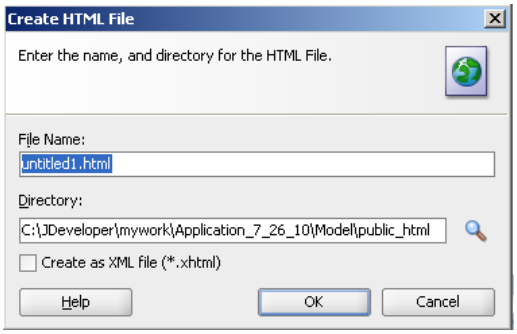
1. In the Applications window, select the project to create the HTML page.
2. Choose **File** > **New > From Gallery** to open the New Gallery.
3. In the **Categories** tree, expand **Web Tier** and select **HTML**.

Figure 16-18 Create HTML Page From the New Gallery



4. Leave the **Directory** field unchanged to save your work in the directory where the system expects to find web application files. In the **File Name** field, enter the name of the file you want to generate then click **OK**.

Figure 16-19 Create HTML Dialog



Using the HTML Core Components

Use the Components window to move commonly used tags into a page and build HTML pages. Tabulated are the commonly used set of HTML JDeveloper element tags.

When you are building your HTML page use the Components window 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, as shown below in the figure and table.

Figure 16-20 Component Palette for HTML Pages

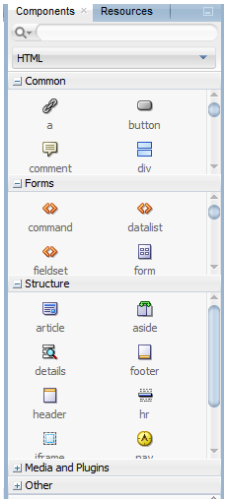


Table 16-16 HTML Common Components

Tag Name	Description
Anchor	Inserts a named anchor <A name> invisible element.
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.
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.

How to Save JSP Files as HTML

JSP files can be saved to HTML by selecting Save as HTML from the File menu in the source editor.

Save your JSP files to HTML using Menu options from the source editor.

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 viewed in a browser it looks like the original HTML page. The saved HTML file can be reopened and viewed as HTML, but no longer understood as code.

Working with HTML Text

Use the Style node in the Properties window to change formatting styles for your HTML text.

How to Add Text to an HTML Page

In the visual editor, type text on HTML pages, and format the text using the toolbar.

Type your text right into your HTML pages in the visual editor.

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>
```

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.

How to Set Text Properties

CSS styles define the formatting for text, but manual HTML formatting overrides the formatting applied by a CSS style. Use the procedure to set text properties.

CSS styles define the formatting for all text in a particular class or redefine the formatting for a particular tag such as h2. 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.

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.

Working with HTML Images

Insert, move, and resize images with the HTML editing tools.

You can insert, move, and resize your images with the HTML editing tools.

JDeveloper supports the following graphic file formats:

- JPEG/JPG
- GIF
- PNG

How to Insert an Image on a HTML File

Insert an image on an HTML page using the Image component in the Components window. Use the procedure to insert an image.

Use the Image component on the Components window to insert an image onto your HTML page.

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 Components window.
   - Drag the Image element from the page of the Components window 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 Applications window. 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 Applications window. Click Y**es** for a Save Image dialog to add the image to the document root. The image will appear on your page.

How to Delete an Image From an HTML File

The keyboard Delete commands delete images.

Delete images using the usual keyboard delete commands.

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.**

How to Resize an Image in an HTML File

Resize images on HTML pages by right-clicking them and using the Properties option.

Right-click and use the Properties selection to resize your images in your HTML pages.

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 Properties window.

Image properties set using the visual editor are marked in the Properties window with a green square. To return a resized element to its original dimensions delete the values in the width and height fields in the Properties window, or click the **Reset Size** button.

How to Use an Image as an HTML Background

Set an image as background on an HTML page using the Properties window. Use the procedure to use an image as background.

Use the Properties window to set your image as a background on your HTML page.

To use an image as a background:

1. Select the page `<body>` element in the Structure window. The Properties window displays the property values for the selected element. If the Properties window is not in view choose **View > Properties window** or Ctrl+Shift+I.
2. Scroll to the background property in the Properties window, 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 Applications window. 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.

How to Move an Image in an HTML File By Dragging

Drag, cut and paste, or use the move command to move images. Images can be moved in the visual editor or the Structure window.

Move images by dragging, cutting and pasting, or using the move command.

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.

Working with HTML Tables

Create and edit tables and data on your HTML pages by using the visual editor. Use the toolbar, the Properties window, and the Structure window to edit tables, to add text, and images.

Use the visual editor to create and edit tables and data on your HTML pages. Use the toolbar, Properties window, and Structure window to 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.

How to Add Text to a Table Cell

Add text, add and remove graphics, or other UI and data elements to tables. Use the procedure to add text to a table cell.

Add text to a table cell by typing it directly in the cell in the editor. Using a table cell as the insertion point, you can add and remove graphics or other UI and data elements to tables.

To add text to a table cell:

1. Click in a cell to add text. 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.

How to Remove Content from One or More Selected Cells

Remove content from table cells using the toolbar menu or backspace key, and a selected row or column can be removed along with the contents of the cell. Use the procedure to remove content from one or more selected cells.

Use the toolbar menu or backspace key to remove content from your cells. Only the contents of the cell, not the cell, will be removed from the table. If the entire row or column is selected, the row or column is removed along with the contents of the cell.

To remove content from one or more cells select cells:

- Click **Delete** or **Backspace**.
- From the main menu select **Edit > Delete.**

How to Format Tables and Cells

Define properties that apply to an entire table, or to selected cells, rows, or columns in the table by using the Properties window. Use the procedure to apply formatting to table cells according to precedence.

Use the Properties window to define properties that apply to the entire table or to selected cells, rows, or columns in the table. When a property such as 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.

How to Set Table and Cell Properties

Set table and cell properties by using the Properties window. Use the procedure to set table and cell properties.

Use the Properties window to set your table and cell properties.

To set table and cell properties:

1. Select the table, row, or cell in the visual editor, or the corresponding <table>, <tr>, and <td> in the Structure window. The Properties window displays the property values for the selected element. If the Properties window is not in view, choose **Window > Properties window** or use the shortcut Ctrl+Shift+I.

   > **Tip:**
   >
   > To quickly locate a property in a long list, click the search button in the Properties window 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**.

How to Set Table and Cell Properties Using the Visual Editor Toolbar

Set table and cell properties using the visual editor toolbar . Use the procedure to set table and cell properties.

Use the visual editor toolbar to quickly set table and cell properties.

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>, and <td> in the Structure window.
2. Use the Properties window to set common table properties such as: align and indent/ outdent, and so forth.

How to Resize a Table

JDeveloper provides user options to resize tables. Use any of the given steps to resize a table.

There are a few choices available to choose from for resizing your tables.

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 Properties window.
- 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.

How to Change the Size of Rows and Columns

Drag rows and columns to resize them in the visual editor. Use the procedure to change the size of rows or columns.

Drag your rows and columns to your size preference in the visual editor.

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.

How to Add Rows or Columns to a Table

Add rows or columns to tables by right-clicking them and using the context menu. Use the procedure to add rows or columns to a table.

Right-click and use the context menu to add rows or columns to your tables.

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**.

How to Remove Rows or Columns in a Table

Remove table rows or columns by right-clicking them and using the context menu Use the procedure to remove rows or columns in a table.

Right-click and use the context menu to remove your table rows or columns.

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.

How to Merge Table Cells

Merge table cells by right-clicking them and using the context menu. Use the procedure to merge table cells.

Right-click and use the context menu to merge your table cells.

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.**

   Alternatively, 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.

How to Split a Table Cell

Split table cells by using the Design option in the main menu. Use the procedure to split a table cell.

Use the Design option in the main menu to split your table cells.

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**.

   Alternatively, 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**.

How to Change the Display Order in a Table Structure

JDeveloper offers several options to modify table structure. Use the procedure to change the display order of rows, columns, or groups of table cells using the visual editor.

There are several options to choose from to modify your table structure.

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.

How to Change the Display Order of Rows Using the Structure Window

Drag rows to a preferred order by using the Structure window. Use the procedure to change the display order of rows.

Use the Structure window to drag your rows to your preferred order.

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.

How to Increase Row or Column Span in a Table

Increase row or column span by selecting options from the context menu in the Structure window. Use the procedure to increase row or column span in a table.

In the Structure window, select and use the context menu to increase your row or column span.

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.

How to Reduce Row or Column Span in a Table

Reduce the span of a table row or column by right-clicking and using the context menu. Use the procedure to reduce row or column span in a table.

Right-click and use the context menu to reduce your table row or column span.

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.

Working with HTML Forms

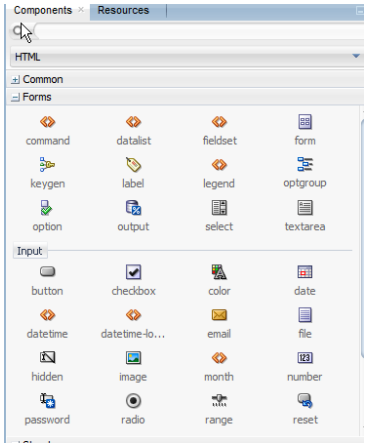
HTML forms on HTML pages help in interacting with users, or gathering information from users of web pages.

Use HTML forms on your HTML pages to interact with or gather information from users of your web pages. Forms are comprised 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.

When you are working on your HTML pages, you can insert your HTML form tags by dragging them from the Components Form window, onto your HTML page. The figure shows the form tags available in the Components window.

Figure 16-21 Components Palette for Forms



How to Create an HTML Form

Create a form element by dragging a Form component from the Components window to an HTML page. Use one of the given options to create a new HTML form.

Drag the Form component from the Components window to your HTML page to create a form element.

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 Components window are added to the page, a <form> element is automatically inserted as a parent element.

To create a new HTML form:

With a 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 Components window.
- Drag the Form element from the Components window to the insertion point. The HTML code to create a skeleton form is inserted. Note that a form appears as a dotted outline in the editor.

How to Delete an HTML Form Element

Remove form elements using the Delete or Backspace keys. Select one of the given options to delete a form element.

Use the usual delete or backspace to remove your form elements.

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**.

How to Insert an HTML Form Field or Button

Insert a form field or button on a page using the Components window. Use the procedure to insert a form field or button.

Use the Components window to insert a form field or button on your page.

To insert a form field or button:

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 field or button to appear, then click the desired element on the Components window.
   - Drag the form field or button element from the Components window to the desired insertion point on the form or Structure window. Note that If you attempt to insert a form field or button without first creating the form, you will get a message “Do you want to add a form element around this component?" Choose **Yes** to automatically create form tags for the field or button. 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.

How to Change the Form Method from the Context Menu

Change a method in a form by right-clicking and using the Form and Method options in the visual editor. Use the procedure to change the form method from the context menu.

Right-click and use the Form selection to change a method in your form.

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.

How to Set Form Processing Information Using the Properties window

Set the values in the property editor for form processing by using the Properties window option from the Window menu. Use the steps to set form processing information.

1. Select the form in the visual editor, or the corresponding <form> element in the Structure window. The Properties window displays the property values for the selected element. If the Properties window is not in view choose **Window > Properties window** 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 Properties window.
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**.

How to Delete a Form Field or Button

You can delete fields or buttons using the main menu or keyboard. Select any of the given options to delete a form field or button, and use the procedure to set form processing information using the Properties window.

Delete fields or buttons using the main menu or keyboard.

To delete a form field or button, do one of the following:

- Select the element and press **Delete** or **Backspace**.
- Select the element and from the main menu choose **Edit > Cut.**

To set form processing information using the Properties window:

1. Select the form in the visual editor, or the corresponding <form> element in the Structure window. The Properties window displays the property values for the selected element. If the Properties window is not in view choose **Window > Properties** 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 Properties window.
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 Properties window toolbar. In the Find text field, type the name of the property, then press Enter.

Working with Cascading Style Sheets

Manage the style and layout of web pages by using Cascading Style Sheets. The tabulated CSS Source editing features help you create and edit CSS files.

Use Cascading Style Sheets (The current standard is CSS2 Revision 1, and we support CSS3 as well) to control the style and layout of your web pages. CSS defines 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 CSS2 R1, see the W3C website at, `http://www.w3.org/TR/1998/REC-CSS2/`.

For more information on CSS3, see the W3C website at, http://www.w3.org/Style/CSS/current-work.en.html.

There are variety of CSS features to help you create and edit you CSS files. Table 16-17 lists the CSS Source features:

Table 16-17 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 Applications window and choose Reformat.
CSS Error handling	Highlights invalid CSS properties, values, and missing semicolon and braces.
Stylesheet linking to HTML files	Links a stylesheet to your HTML files simply by dropping a Link element into your HTML page. Another option is to choose **CSS** in the Components window. The list of available CSS files displays in the Components window. You can then drag and drop any CSS file from the Components window to the page.
Style preview	Allows you to see what your styles look like while you're coding.
Code colors	Helps you 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.
ADF Skin Editor	Creates and modifies ADF skins. ADF skin is a type of CSS file that applies to an ADF application.
Drag and Drop Linking	Links a CSS file to an HTML or JSP page.
Quick docs	Opens the description from the W3C standard.

Selecting and Grouping CSS Elements

When you open a CSS file for editing, the Structure window displays CSS selectors such as Elements, Classes, and IDs.

When a CSS file is open for editing, CSS selectors are displayed in the Structure window using icons:

Elements

Element is 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;.}`

Classes

Class represents 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.

IDs

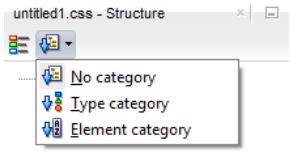
ID represents a style unique to one HTML element. For example `p#para1 {color:green;}` defines the p element that has the id `value="para1" and *#ver905 {background-color:red;}` defines the first HTML element with id `value="ver905"`.

Working with Grouped Elements

The Categories list in the Structure window toolbar shows CSS selectors by categories, No Category, Type Category, and Element Category. The Separate Grouped Selectors option can be used to separate or ungroup the selector categories.

You can use the Categories dropdown list in the Structure window toolbar to show CSS selectors by categories, as shown in the figure.

Figure 16-22 CSS Selector For Sorting Options



There are three group types from which to choose

No Category

When No Category is selected, your Structure window display is in the order of appearance in the CSS file. This is the default setting.

Type Category

When you select Type Category, your Structure window display is arranged by the CSS selector types of element, class or ID.

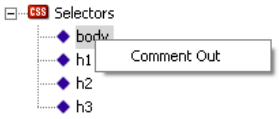
Element Category

When you select Element Category, your Structure window display is arranged by HTML element or tag.

Select a CSS selector in the Structure window to highlight the selector in the editing window and edit associated properties and values in the Properties window.

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.



How to Create a Simple Cascading Style Sheet

Create a simple Cascading Style Sheet page using New Gallery. Use the procedure to create a simple Cascading Style Sheet.

Use the New Gallery to create your simple CSS page.

To create a simple Cascading Style Sheet:

1. In the Applications window, select your project.
2. Choose **File > New > From Gallery > HTML > CSS File.**
3. Leave the **Directory Name** field unchanged to save your work in the directory where your web application files are.
4. In the **File Name** field, enter the name of the file to generate then click **OK**. A simple CSS file is generated and appears in your active project in the CSS folder under Web Content.

How to Set or Modify CSS Selector Properties and Values

Set your Cascading Style Sheet properties and values using the Properties window. Use the procedure to set or modify CSS selector properties and values.

Use the Properties window to set your CSS properties and values.

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 to set a property.
2. In the Properties window, scroll until the property is visible. To quickly locate a property in a long list, click the search button in the Properties window 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.**

How to Format Text with CSS Properties

Although you can use CSS styles, and manual or online HTML formatting within the same page, manual HTML formatting overrides CSS style formatting. Use the procedure to format CSS text properties.

You can also use 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 class or redefines the formatting for a particular tag such as h2.

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.

To format CSS text properties:

1. Select the text to format.
2. Use the Font field in the Properties window to set text properties.

How to Edit a CSS File

Set and modify CSS properties using the CSS Source Editor. Use the procedure to edit a CSS File in the Source editor.

Use the CSS Source Editor to set and modify your CSS properties.

To edit a CSS File in the Source editor:

1. In the Applications window, double-click the CSS file to open it in the default Source editor window.
2. Enter the CSS selector (HTML element, class, or ID) 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:`

   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;`

   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;`
5. 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.

Working with Java Server Pages

JDeveloper supports tools for user interface development using the JavaServer Faces technology within the Java EE platform.

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`.

Building Your JSP Application

Application development starts with the creation of an application template for web pages.

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. Start by using the application project templates. Choose from a combination of options in the New Gallery.

JSP Core Components

JDeveloper provides a Components window with standard JSP components that can be used to build JSP pages. Tabulated are the JSP core components.

JDeveloper comes with a Components window with standard JSP components that you can easily drag and drop onto your JSP pages, as shown in Figure 16-23 and Table 16-18.

Figure 16-23 JSP Core Components Window

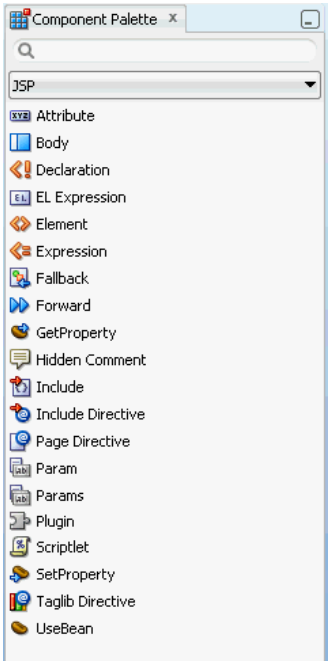


Table 16-18 JSP Core Components

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.
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.
Params	Provide key value information.
Plugin	Executes an application 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.

How to Create JSP Pages

When you create the web pages for an application, a JSP is generated and included in the active project along with the deployment descriptor file `web.xml` that is used by the Integrated WebLogic Server. Use the procedure to create a new JSP page.

The New Gallery wizard walks you through all of the necessary steps to build the web pages for of your application. After you create the page, 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.

To create a new JSP page:

1. In the Applications window, 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**.

How to Register a Servlet Filter in a JSP Page

Create filters to process requests or responses to or from a JavaServer Page using the Create Servlet Filter wizard available from the Web Tier category in New Gallery. Use the procedure to register a servlet filter on 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.

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.

To register a servlet filter in a JSP page:

1. In the Applications window, 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**.

Understanding Flow Control in JSPs

Flow control in web applications direct the display content of the web browser in response to specific user actions. JDeveloper supports several flow control approaches such as

JavaServer Faces page navigation, the ability to visually design page flows, and all-in-one JSP page development.

Web applications implement flow control by directing the display content of the web browser in response to specific user actions. Generally it makes sense to create separate JSP pages or sets of pages for each task in the workflow. 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.

The following flow-control approaches are supported in JDeveloper:

- 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.

How to Handle JSP Flow Control

JDeveloper supports various options for the implementation of JSP page flow control that determine how applications handle Request objects.

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.
- You can cleanly separate JSP pages and their actions by implementing the controller outside of the JSP page.

Debugging and Deploying JSPs

JDeveloper supports the deployment of web applications on any Java EE application server by means of the creation of Web Module Archives.

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.

How to Debug a JSP

Debugging JSP pages translates the JSP into a servlet, starts the Integrated WebLogic Server process, and results in invoking the JSP in the default Web browser. Use the procedure to debug a JSP page.

You can debug your JSP pages in a couple of clicks.

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 debug a JSP:

1. In the Applications window, 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.

How to Create a Web Deployment Descriptor

Create a web deployment descriptor using New Gallery, or by adding a JSP page. Use the procedure to create a web deployment descriptor.

The are a few ways to create a web deployment descriptor. You can use the New Gallery, or add a JSP page. New deployment descriptors are added to the WEB-INF folder in the project, and it will be opened in an XML editor window.

To create a web deployment descriptor:

1. In the Applications window, select the project for which you want to create a web deployment descriptor.
2. Add a JSP file to the project. The web.xml file is added 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 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.

How to Edit Web Deployment Descriptor Properties

Edit descriptor properties by right-clicking and opening the properties dialog. Use the procedure to edit web deployment descriptor properties.

Right-click to open the properties dialog and edit your descriptor properties.

To edit web deployment descriptor properties:

1. In the Applications window, 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.

How to Edit a Web Deployment Descriptor as an XML File

The descriptor file in XML format can be edited by opening the file in the XML source editor or the Overview editor. Use the procedure to edit a web deployment descriptor as an XML file.

The descriptor file is in XML format in the WEB-INF folder. You can open the file in the XML source editor or the Overview editor.

To edit a web deployment descriptor as an XML file:

1. In the Applications window, select the web deployment descriptor in the `WEB-INF` folder.
2. Right-click and choose **Open.** The file opens in an XML editor.

Running a JSP

The Integrated WebLogic Server manages the running of JSPs in JDeveloper.

The Integrated WebLogic Server handles running your JSPs in JDeveloper. You can run your JSP pages right from the page.

How to Run a JSP

In JDeveloper, you can build a JSP, run, and edit it. Use the procedure to run a JSP.

After building your JSP, you can run it in a couple of clicks. You can also edit your JSP while you are running it.

To run a JSP:

1. In the Applications window, 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.

Several things automatically happen when you run your JSPs:

- The JSP is translated into a servlet and compiled.
- The resulting classes are run directly from the output project directory.
- The web.xml file is modified to include the servlet name and class information.
- The JSP is invoked 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 JSP Files While They are Running

Without a WebLogic Server restart, run JSP files, and modify and view the changes made to them.

You can modify and view changes that you make to your JSP files as they are running, without having to restart WebLogic Server. To view changes in your browser, you can either reload the page from the browser or run the page again.

Running JSPs with ADF Business Components Application Modules

If JSPs run with business components application modules in both the Integrated WebLogic Server and in a remote server instance, declare that the middle tier runs inside a WebLogic Server instance with the `jbo.server.in_wls=true` property.

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 need to 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

If the timestamp of a source JSP file changes to an earlier timestamp, the JSP must be forced to recompile using one of the rebuild options.

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`.

Understanding JSP Segments

JSP fragments are JSP pages that can be included in other JSP pages. Being incomplete pages, the fragments need to use the `.jspf` extension.

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.

Developing Applications with Java Servlets

Servlets are platform-independent, server-side Java components that are used to extend the capabilities of a web server. Servlets help dynamically tailor content, function, and the appearance of web pages.

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, annotations and deployment descriptors, web fragments for pluggability and extensibility, and asynchronous support (Servlet API 3.0). 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 *Developing Web Applications, Servlets, and JSPs for Oracle WebLogic Server*.

Understanding Servlet Support in JDeveloper

HTTP servlets process client HTTP requests, and provide dynamic content by returning to clients the results of database queries.

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.
- 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.

What You May Need to Know About Servlet Filters

Filters do not usually create responses, but are reusable pieces of code that modify the content of HTTP requests, responses, and header information. Create filters using the Create Servlet Filter wizard available from the Web Tier Servlets category in New Gallery.

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.

What You May Need to Know About Servlet Listeners

Listeners define listener objects whose methods get invoked when life cycle events occur and thus monitor and react to events on a servlet life cycle. Create listeners by using the Create Servlet Listener wizard available from the Web Tier, Servlets category in New Gallery.

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.

How to Generate an HTTP Servlet

HTTP servlets process client HTTP requests, and provide dynamic content by returning to clients the results of database queries. Use the procedure to generate the HTTP Servlet.

JDeveloper will create the servlet for you based on information you specify, including the methods and parameters for the servlet.

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.

Note that the deployment descriptor file will take precedence over any annotations you have made to your servlet. For example, while you can create a class with an annotation such as `@Servlet`, the deployment descriptor file will override it if the values are different. Be sure to

reconcile any differences between the annotations and their declarations in the deployment descriptor file.

To generate the HTTP Servlet:

1. In the Applications window, select the project in which you want to create the new servlet.
2. From the main menu, choose **File > New > From Gallery > Web Tier > Servlets**, or right-click and choose **New**. The New Gallery opens.
3. In the Items list, double-click **HTTP Servlet** to launch the Create HTTP Servlet wizard.

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. The Java class contains an initialization method and the HTTP methods that were specified, and the methods must be implemented to customize the 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.

How to Use the HTTPServletRequest Object

In a basic servlet method, the first HTTP argument is an `HttpServletRequest` object that provides methods to access the HTTP header data, the HTTP request method, and the client arguments.

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 16-19 summarizes the relationship between the possible kinds of HTTP requests and the corresponding methods you should use when implementing your servlet methods.

Table 16-19 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.

How to Use the HTTPServletResponse Object

The second HTTP argument in a basic servlet method is an `HttpServletResponse` object that encapsulates the information from the servlet to be returned to the client. Change the HTTP Response Type using Generate Content Type that is available in the Create HTTP Servlet wizard.

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();
  }
}
```

How to Create a Servlet Filter

In a basic servlet method, the first HTTP argument is an `HttpServletRequest` object that provides methods to access the HTTP header data, the HTTP request method, and the client arguments.

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.

The Create Servlet Filter wizard creates the servlet filter for you based on information you specify, including the implementation class and initialization parameters.

There is an option when you enter servlet details for your new servlet filter to select the registration vehicle for your servlet. You can either add a servlet entry to the Configuration file, or select Annotations to create annotations in the source code.

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.

Note also that you can use the annotation `@ServletFilter` to declare a filter class.

To create a servlet filter:

1. In the Applications window, 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 > From Gallery > Web Tier > Servlets** .
3. In the **Items** list, double-click **Servlet Filter** to open the Create Servlet Filter wizard.

How to Create a Servlet Listener

Specify information such as the implementation class and interface, and create a servlet listener using the Create Servlet Listener wizard. Use the procedure 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.

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.

The Create Servlet Listener wizard creates the servlet listener for you based on information you specify, including the implementation class and interface.

There is an option when you enter servlet details for your new servlet filter to select the registration vehicle for your servlet. You can either add a servlet entry to the Configuration file, or select Annotations to create annotations in the source code.

The annotation @ServletContextListener, added to a class definition, can also be used to declare a class of servlet listener:

```
@ServletContextListener
```

```
public class MyListener {

public void contextInitialized (ServletContextEvent sce) {


}

.....

}
```

To create a servlet listener:

1. In the Applications window, 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 > From Gallery > Web Tier > Servlets**.
3. In the **Items** list, double-click **Servlet Listener** to open the Create Servlet Listener wizard.

Registering a Servlet Filter in a JSP Page

Create a new filter to process requests or responses to or from a JavaServer Page by using the Create Servlet Filter wizard available from the Web Tier category in New Gallery. Use the procedure to register a servlet filter on 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.

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.

To register a servlet filter in a JSP page:

1. In the Applications window, 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 your servlet information and click **Next**.
7. Select **Map to Servlet or JSP**, and select the name of the JSP from the dropdown list. Then click **Next**. Alternatively, select **Map to URL Pattern**, then select a URL pattern to which the filter will map, to associate it with groups of servlets or static content.
8. Click **New**, and enter the name and value for each initialization parameter. Then click **Finish**.

How to Run a Servlet

The Integrated WebLogic Server runs servlets, Java programs that run in a Java EE application server, in JDeveloper. Use the procedure 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 Applications window, 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
  ```

How to Debug a Servlet

Debug a servlet using the embedded Integrated WebLogic Server in JDeveloper. Use the procedure 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 Applications window 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
  ```

How to Deploy a Servlet

Create a Web Module Archive to deploy Servlet applications on any Java EE application server.

JDeveloper supports deploying your Servlet applications on any Java EE application server through the creation of a Web Module Archive (WAR). See *Developing Web Applications, Servlets, and JSPs for Oracle WebLogic Server*.

Developing Applications with Script Languages

JDeveloper supports script languages, JavaScript and JSON, and script developer tools including code insight, breadcrumb support, and the JDeveloper structure pane. JDeveloper also provides Audit support, Quick JsDoc support on JavaScript identifiers, Code hover support, brace matching, and other editing support.

JDeveloper provides scripting functionality, including support for basic JavaScript when working with JSP and HTML pages.

JDeveloper supports script languages, specifically JavaScript and JSON, through the 1.8.1 and 1.8.5 standards, which include native support for JSON encoding and decoding. JDeveloper offers the script developer tools including 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.

JDeveloper also provides the following kinds of support during script language development:

- Audit support . Warnings are shown as yellow lines and errors as red lines in the code editor. Also, warnings and errors are indicated on the right hand side in the file overview margin.
- Quick JsDoc support on JavaScript identifiers. This is similar to the quick JavaDoc available when developing in Java.
- Code hover support. Pressing the shift key over JavaScript identifiers displays the definition in a ghost popup pane.
- Brace matching and all other editing support, as available in other languages like Java.

How to Work with JavaScript Code Insight

When you type inside a script region, or inside an HTML event handler, JavaScript Code Insight in JDeveloper completes labels, variables, parameters and functions. In the editing pane, the feature displays a dynamic list of possible completions for a given JavaScript function.

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 global 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.

The JavaScript Code Insight feature creates code 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. Furthermore, JavaScript Code Insight supports the HTML 5 standard.

How to Use Breadcrumb Support

JDeveloper displays a breadcrumb trail showing the position of a function in the JavaScript hierarchy when you edit a JavaScript file in the Source Editor and place the mouse pointer in a function.

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.

Working with Script Languages

JDeveloper supports script languages that allow the use of script elements in an HTML or JSP page, or the use of references to script files which are associated with an application. Use the syntax highlighting feature in the code editor, or create JSON files.

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.

How to Create a Script

Create a client-side script for inclusion or embedding in an HTML or JSP page. Use the procedure to create a script in JDeveloper.

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 Applications window.
2. In the Components window, 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>
```

How to Add Script Language Elements to an HTML or JSP Page

Basic JavaScript support is available when working with HTML and JSP pages. Use the procedure to insert JavaScript into a JSP or HTML 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 Applications window 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 Applications window, drag a JavaScript onto the page where appropriate. If you drag a JavaScript from the Components window, 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.

How to Set Syntax Highlighting

When editing Java, JavaScript, and JSON files, syntax highlighting helps you easily identify syntax elements. Use the procedure to set syntax highlighting options for JavaScript in the Code Editor.

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.

How to Associate JavaScript File Extensions

JDeveloper recognizes files with the `.js` file extension, the default, as JavaScript. Use the procedure to add or remove file extensions for JavaScript file recognition in JDeveloper.

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.

How to Create a JSON File

Create a JSON file that allows the passing of structured data between applications or between files within an application, Use the procedure 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.

How to Use Structure Pane Support

JDeveloper tracks the location of edits to a JavaScript file in the structure of the project or application you are building, and displays it in the Structure Pane. Use the step to find a location in the code editor from the Structure Pane.

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.

Refactoring JavaScript Code

JDeveloper supports refactoring or the renaming of references to a function or variable, which modifies code structure without altering program behavior. .JDeveloper supports refactoring operations for JavaScript code and files such as renaming references to a function, variable, or label, and safe deletion.

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.

Finding Usages of Code Elements

Search within a JavaScript file for specific usages of code elements so that you can locate and change them, if needed. Use the procedure to search in a JavaScript file for a function, variable, or label.

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 Comments for Textual Usages
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.

> **Note:**
>
> The Log Window is not displayed by default when you start JDeveloper. To display the Log Window, select **Windows > Log**.

Renaming a JavaScript Code Element

In JavaScript code, rename the definition and all references to a function or variable, the scope of a renaming operation being the full scope of the element in the project. Use the procedure to rename a 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.
   - 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**.
   - 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 Comments for Textual Usages** 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.

Deleting a JavaScript Code Element

Delete the definition of a function, label or variable, and if the element is in use, the delete action must be confirmed. Use the procedure to delete a 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.
   - 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**.
   - 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.
   - Click **OK** to delete the element's definition.

How to Preview a Refactoring Operation

Inspect and resolve usages in the Preview Log window before you confirm the modifications, and commit the operation. Use the steps to view a usage in an Edit window, exclude a usage from the refactoring operation, and to commit the 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.

In the case of a very lengthy refactoring operation (for example, one involving many calls to the same JavaScript function in a long source file), JDeveloper displays the processing status in the status bar (below the Log window).

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.

How to Reformat JavaScript Code

Reformat JavaScript code, line up parallel elements, and find visual cues to the parts of the function that is being edited. Use the procedure to reformat a section of 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.

How to Change Code Formatting Preferences

Customize the code editor appearance, general behavior, and Code Insight and Java Insight options. Use the step 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.

How to Use Code Folding

Reformat a `.js` file if you have made modifications that affect readability or file size by folding selected logical elements of the file. Use the steps 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 +.

- Click on the + sign to unfold the code, displaying the full contents of the area you previously folded.

  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.

How to Refactor and Move a File

JDeveloper updates references to a changed file in the application using the `<script src=...>` tag. Use the procedure to refactor and move a JavaScript function.

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 Applications window 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.

Working with JSP and Facelet Tag Libraries

JDeveloper supports JSP and Facelet custom tag libraries to develop reusable modules called custom actions. Custom actions perform tasks such as form processing, accessing databases or email, and controlling flow.

JDeveloper supports JSP 2.0, 2.1, and1.2, as well as 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, add 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. The taglib.xml file is the document that describes your facelets tags.

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.

Using Tag Libraries with Your Web Pages

Custom tag libraries that are supported by JDeveloper Code Insight simplify the creation of new JSP or facelet custom tag libraries, and the import and registering of custom tag libraries. Use the JSP Tag Libraries option or Facelets Tag Libraries option available in Manage Libraries for Tools.

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 Components window. 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 Components window to display the available tags on the Components window 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**.

For a complete list of included tag libraries see the JDeveloper Tag Library Reference under the Javadoc and Tag Library Reference node in the Help Table of Contents.

Tag support includes these standard 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 2.1.** 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://docs.oracle.com/javaee/6/javaserverfaces/2.1/docs/vdldocs/facelets/`
- **Trinidad Components 2.0.** For more information, see the Apache Trinidad page at: `http://myfaces.apache.org/trinidad/index.html`.
- **ADF Faces Components 12**. For more information, see the *Tag Reference for Oracle ADF Faces*.

How To Add, Delete or Edit Project Level Tag Libraries

Manage libraries from the Application option on the main menu. Use the procedure to add, delete, or edit project level tag libraries.

Manage your libraries from the Application option on the main menu.

To add, delete, or edit project level tag libraries

1. Choose **Application > Project Properties > JSP Tag Libraries**
2. Add, delete, or edit project tag libraries as necessary.

How to Browse to a JSP Tag Library Descriptor (TLD) File

On a JSP page, right-click and use the tag library browse option to go to a JSP tag library descriptor file. Use the procedure to browse to a JSP tag library descriptor file.

You can right-click from anywhere on a JSP page to get to the tag library browse option.

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.

How to Browse Pages or Individual JSP Tags

Browse tags and pages from the Configure Palette option from the Tools option on the main menu.

How to Work with Custom Tag Libraries

Create the tag library descriptor file, and then create simple tags or component tags to create a custom tag library.

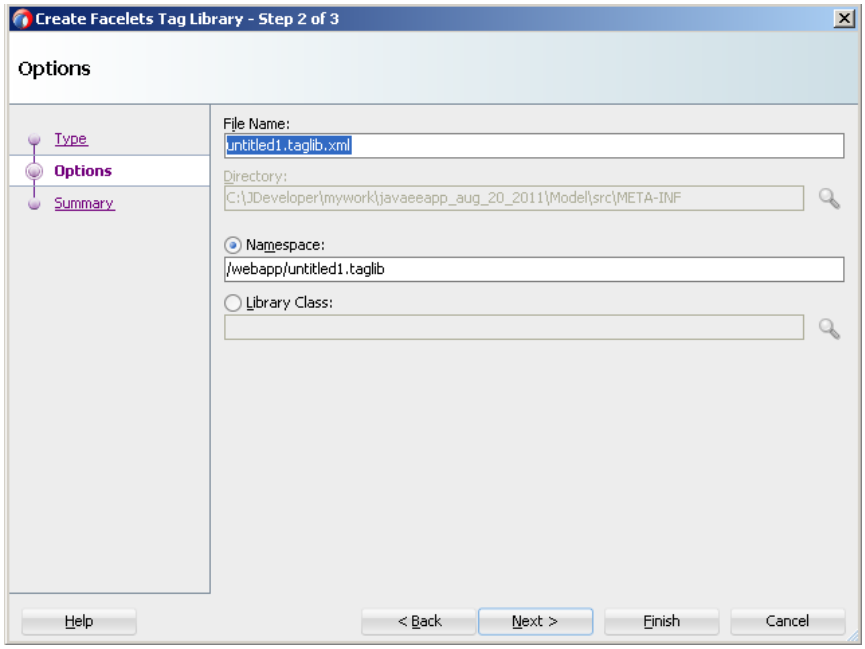
To create a custom tag library, create the tag library descriptor file and then create simple tags or component tags. A tag library descriptor file (TLD) is an XML document that describes your tag library and each tag in it. It is used by a container to validate the tags. Once you create tags, you can add attributes and scripting variables to them.

How to Create a Custom JSP or Facelets Tag Library

Use the procedure to create the new tag library. Create custom tags, either project based or deployable library type using the New Gallery tag library wizards.

Use the New Gallery tag library wizards to create your custom tags. You can choose to create a project based or deployable library type, depending on whether you want to share the library or keep it within one project. The second step in the wizard lets you name and add location details for you tag library, as shown in the figure.

Figure 16-24 Facelets Tag Library Wizard



To create a custom JSP or facelets tag library:

1. In the Applications window, select the project in which you want to create the new tag library.
2. Choose **File > New** > **From Gallery** 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**.

How to Add your Custom Tag Library to the Components Window

Use the procedure to add an unregistered custom tag library to the Components window. Make available the registered custom tag libraries available in the Components window of a projects using the Add option for JSP Tag Libraries/Facelet Tag Libraries available for Project Properties in the Application menu.

To make your registered custom tag libraries available in the Components window of your projects, go to **Application > Project Properties > JSP Tag Libraries/Facelet Tag Libraries > Add**. Check **Show Tag Library in Palette**.

To register a deployed library, see How to Register a JSP or Facelet Tag Library

When you create a custom tag library with the New Gallery wizard, it is added to the Components window only if you select the **Project Based** option in the **Create Facelets Tag Library** wizard. If you select, or leave the default choice of **Deployable** at creation time, you will need to perform additional steps to register your library and add it to your project through the Project Properties Add Tag Libraries feature. The tag library will then show in your Components window.

To add a unregistered custom tag library to the Components window:

1. Deploy your new custom tag library.
2. Register your deployed library. After your library is registered it shows up in the Project Properties and Add Tag library options.
3. Add the tag library to your project. The library now shows in your Components window.

How to Register a JSP or Facelet Tag Library

Tag libraries must be registered. Use the procedure to register a JSP or facelet custom tag library.

Register your tag libraries to tell facelets that they exist.

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. Select the User folder and 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.

How to Edit a TLD File in the XML Source Editor

You can edit a TLD file and change element attributes in the XML Source Editor. Use the procedure to edit a TLD file.

You can edit your TLD file and change your element attributes.

To edit a TLD file in the XML Source Editor:

1. In the Applications window, 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 Components window, choose **View > Components window** to open the Palette and select **Tag Lib** or one of the available pages from the dropdown list. Then choose elements from the page.

How to Add a Scripting Variable to a Tag

Scripting variables are the variables that are available to the JSP page when the page is called. Use the procedure to add a scripting variable to a tag.

Scripting variables are variables that are available to the JSP page when any JSP page is called. Scripting variables may be any scripting variables but when you are dealing in reference of JSP page it means page level variables declared by the JSP page. You can access scripting variables in scriptlet, declaration, expressions.

To add a scripting variable to a tag:

1. In the Applications window, 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.

How to Deploy Your Custom JSP/Facelets Tag Library as a JAR File

Deploy custom tag libraries using the deploy to context menu. Use the procedure to deploy a custom JSP tag library or a facelets tag library as a JAR File.

Use the deploy to context menu to deploy your custom tag libraries.

To deploy your custom JSP tag library or facelets tag library as a JAR File:

1. In the Applications window, 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.

17

Developing with EJB and JPA Components

This chapter describes how to use JDeveloper tools to build the business tier of a Java EE enterprise application using Enterprise JavaBeans (EJB) 3.x and Java Persistence API (JPA) components.
This chapter includes the following sections:

- About Developing with EJB and JPA Components
- Support For EJB Versions and Features
- Building EJB 3.x Applications and Development Process
- How to Work with an EJB Business Services Layer
- Using Java EE Design Patterns in Oracle JDeveloper
- Using Java EE Contexts and Dependency Injection (CDI)
- Building a Persistence Tier
- Implementing Business Processes in Session Beans
- Modeling EJB/JPA Components on a Diagram
- Deploying EJBs as Part of an Web Application
- Deploying EJB Modules and JPA Persistence Units
- Running and Testing EJB/JPA Components

About Developing with EJB and JPA Components

JDeveloper provides wizards to create EJB projects, entities, persistence units, session beans, and message-driven beans.

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.

Support For EJB Versions and Features

JDeveloper supports EJB 3.x. Listed are the features supported by EJB 3.1 specification and the differences between EJB 3.0 and previous versions.

JDeveloper supports EJB 3.x, as well as versions 1.0 through 2.1. However, the EJB wizards do not support the creation of EJBs earlier than version 2.x, but will instead prompt you to import such older EJBs into version 3.1. The current JDeveloper documentation, including this chapter of the *User Guide* and the embedded online help, focus on EJB 3.1 development tasks.

> **Note:**
>
> Previous versions of the JDeveloper 10*g* documentation tell how to work with EJB 2.1 and earlier. 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 EJB 3.1 specification and documentation, refer to the "Java Platform, Enterprise Edition (Java EE) Technical Documentation" page at `http://docs.oracle.com/javaee/`.

> **Note:**
>
> If you are using EJB 3.x, you may be using annotations instead of some deployment files. Include deployment descriptors to override annotations or specify options not supported by annotations.

Supported New EJB 3.1 Features

The EJB 3.1 specification provides simplified programming and packaging model changes.

- **Singleton Session Bean** – Singleton session beans provide a formal programming construct that guarantees a session bean will be instantiated once per application in a particular Java Virtual Machine (JVM), and that it will exist for the life cycle of the application. With singletons, you can easily share state between multiple instances of an enterprise bean component or between multiple enterprise bean components in the application.
- **Simplified No Interface Client View** – The No-interface local client view type simplifies EJB development by making local business interfaces optional. A bean that does not have a local business interface exposes a no-interface view, which provides the same enterprise bean functionality without having to write a separate business interface.
- **Packaging and Deploying EJBs Directly in a WAR File** – EJB 3.1 provides the ability to place EJB components directly inside of Web application archive (WAR) files, removing the need to produce archives to store the Web and EJB components and combine them together in an enterprise application archive (EAR) file.
- **Portable Global JNDI Names** – The Portable Global JNDI naming option in EJB 3.1 provides a number of common, well-known namespaces in which EJB components can be registered and looked up from using the pattern `java:global[/<app-name>]/<module-name>/<bean-name>`. This standardizes how and where EJB components are registered in JNDI, and how they can be looked up and used by applications.
- **Asynchronous Session Bean Invocations** – An EJB 3.1 session bean can expose methods with asynchronous client invocation semantics. Using the `@Asynchronous` annotation in an EJB class or specific method will direct the EJB container to return control immediately to the client when the method is invoked. The method may return a future object to allow the client to check on the status of the method invocation, and retrieve result values that are asynchronously produced.
- **EJB Timer Enhancements** – The EJB 3.1 Timer Service supports calendar-based EJB Timer expressions. The scheduling functionality takes the form of CRON-styled schedule definitions that can be placed on EJB methods, in order to have the methods be automatically invoked according to the defined schedule. EJB 3.1 also supports the

automatic creation of a timer based on metadata in the bean class or deployment descriptor, which allows the bean developer to schedule a timer without relying on a bean invocation to programmatically invoke one of the Timer Service timer creation methods. Automatically created timers are created by the container as a result of application deployment.

Supported New and Changed EJB 3.0 Features

The key differences between EJB 3.0 and previous versions are:

- **Simplified EJBs** - EJB 3.0 eliminated 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 was simplified, and dependency injection is used instead. Metadata annotation is used for dependency injection.
- **Enhanced Life-cycle Methods and Callback Listener Classes** - Unlike previous versions of EJB, you do not have to implement all unnecessary callback methods. Instead, you designate any arbitrary method as a callback method to receive notifications for life-cycle 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.
- **Life-cycle 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.
- **Life-cycle 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 simplified 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 introduced 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 simplified 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 improved 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.
- **Life-cycle Methods and Callback Listeners** - Callback listeners are supported with entity beans. Callback methods are either specified using annotations or a deployment descriptor.

Building EJB 3.x Applications and Development Process

Use the JDeveloper features to create the framework for an EJB web application, set up the EJB business layer, and use the wizards to create entities, session beans and facades, and build a persistence unit. Enable data controls and test them using the JDeveloper integrated server capabilities.

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.

EJB 3.x Application Development Process

JDeveloper includes tools to create entities, create session beans and facades, deploy EJBs, test EJBs remotely, and register business services with data controls.

JDeveloper includes tools for developing EJB applications, as described in the following sections.

- Creating Entities
- Creating Session Beans and Facades
- Deploying EJBs
- Testing EJBs Remotely
- Registering Business Services with Data Controls

Creating Entities

Create entities using the entity wizards, or create entities from tables using online, offline, or application server data source connections.

Use the entity wizards to create entities or to create entities from tables using online, offline, or application server data source connections. Use the Entities from Tables wizard to reverse-engineer entities from database tables. In the entity wizards you can select or add a persistence unit and a database connection, or you can select a database to emulate. You can also select database tables for your entity. See 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 using O-R mapping metadata, you have more control over the mapping, but you must code the annotations by hand. SeeMetadata Annotations for O-R Mapping .

Creating Session Beans and Facades

Create a session bean, and then generate session facade methods for entities in the project. A session facade aggregates and presents data, and provides a location for business logic.

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. See Implementing Business Processes in Session Beans and 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 Applications window and choosing **Edit Session Facade**.

Deploying EJBs

The Oracle WebLogic Server that JDeveloper provides serves as a container for deployed EJBs. Either use the WebLogic-specific deployment profile that JDeveloper generates, or create a custom profile..

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. See Deploying EJB Modules and JPA Persistence Units .

Testing EJBs Remotely

Test EJBs remotely using the sample client that JDeveloper creates for use with a remote server.

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. SeeHow to Test EJB/JPA Components Using a Remote Server.

Registering Business Services with Oracle ADF Data Controls

Use ADF components to enable data controls for entities.

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. See Using ADF Data Controls.

How to Work with an EJB Business Services Layer

Create a model business services layer for a web-based EJB 3.x application. Use the step to create a web-based application, and use the procedures to create JPA entities, implement a session facade, register the business services model project with the data control, run, test, and deploy the application.

Create a model business services layer for a web-based EJB 3.x application.

To create a web-based application:

- Select **File > New > Application**.

  The General category in the New Gallery provides a list of available applications. For EJB projects you can choose to build either a custom application or the Java EE Web application. The Java EE Web Application creates an EJB/JPA data-bound web application.

  Tip: Frequently-used selections are automatically saved to the New menu for easy access.

To create JPA entities:

1. In the Applications window, right-click the project in which you want to create a JPA entity and choose **New**.
2. In the New Gallery, expand **Business Tier**, select **EJB** and then select **Entity** or **Entities from Tables** and click **OK**.

   Tip: Frequently-used selections are automatically saved to the New menu for easy access.

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 Entity or Entities from Tables wizard to create JPA entities.

   For more information at any time, press **F1** or click **Help** from within the wizard.

To implement a session facade:

1. In the Applications window, right-click the project in which you want to create a session facade and choose **New**.
2. In the New Gallery, expand **Business Tier**, select **EJB** and then select **Java Service Facade (JPA/Toplink)** and click **OK**.

   Tip: Frequently-used selections are automatically saved to the New menu for easy access.
3. Follow the steps in the Java Service Facade 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.

   For more information at any time, press **F1** or click **Help** from within the wizard.

To register the business services model project with the data control:

- Right-click your session bean in the Applications window 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 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.

Using Java EE Design Patterns in Oracle JDeveloper

The Java EE design patterns provide best practices for application design. EJB applications can implement the MVC, Session Facade, and the Business Delegate Java EE design patterns.

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. See 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. See "Using ADF Data Controls" in *Developing Applications with Oracle ADF Data Controls*.

Using Java EE Contexts and Dependency Injection (CDI)

Use Contexts and Dependency Injection services for the Java EE platform, such as Contexts, Dependency Injection, and Interceptors, when enterprise beans are used with JavaServer Faces technology in web applications.

Contexts and Dependency Injection (CDI) for the Java EE platform is a set of services that, used together, make it easy for developers to use enterprise beans along with JavaServer Faces technology in web applications.

The most fundamental services provided by CDI are as follows:

- **Contexts:** The ability to bind the life cycle and interactions of stateful components to well-defined but extensible life cycle contexts.
- **Dependency Injection:** The ability to inject components into an application in a typesafe way, including the ability to choose at deployment time which implementation of a particular interface to inject. Dependency injection is used either from stateful or stateless session beans, as well as with message-driven beans. Developers can use either metadata annotations or deployment descriptors to inject resources, EJB context, or environment entries used by EJBs.
- **Interceptors:** 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.

For more information about CDI, see "Introduction to Contexts and Dependency Injection for the Java EE Platform" in the *Java EE 6 Tutorial*.

The following CDI features are surfaced in JDeveloper for EJB 3.x:

- beans.xml File
- Interceptor Binding Type
- Qualifier Type
- Scope Type
- Stereotype

beans.xml File

Any application that uses CDI (Contexts and Dependency Injection) must have a `beans.xml` file. If an application is part of an EJB project (based on the project's `.jpr` file), the `beans.xml` file is generated in the project's META-INF directory. For all other application types, such as Web applications, the `beans.xml` file is generated in the project's WEB-INF directory. The `beans.xml` file can be empty.

For more information about managed beans, see "About Beans" in the *Java EE 6 Tutorial*.

To create a beans.xml file:

1. Select a project in the Applications window and from the File menu, choose **New > From Gallery**.
2. From the New Gallery, expand **Business Tier** and **Contexts and Dependency Injection**, and then select **beans.xml (Contexts and Dependency Injection)** and click **OK**.
3. In the Applications window, browse to the Web Content folder and double-click the `beans.xml` file to open it in the Java source editor.
4. Use the following options on the general tab to define one or more of the following child elements in the `beans.xml` file:
   - **interceptors** – Interceptors are used to perform cross-cutting tasks, such as logging or auditing, that are separate from the business logic of the application and which are repeated often within an application. Interceptors allow you to specify the code for these tasks in one place for easy maintenance. For more information about interceptors, see "Using Interceptors in CDI Applications" in the *Java EE 6 Tutorial*.
   - **decorators** – Decorators are outwardly similar to interceptors; however, they actually perform business logic by intercepting business methods of beans. This means that instead of being reusable for different kinds of applications as are interceptors, their logic is specific to a particular application. For more information about decorators, see "Using Decorators in CDI Applications" in the *Java EE 6 Tutorial*.
   - **alternatives** – When you have more than one version of a bean that is used for different purposes, you can choose between them by using alternatives. For example, you might have a full version of a bean and also a simpler version that you use only for certain kinds of testing. For more information about alternatives, see "Using Alternatives in CDI Applications" in the *Java EE 6 Tutorial*.

Interceptor Binding Type

Interceptor bindings are Java annotations that associate an interceptor with any managed bean that is not itself an interceptor or decorator, or with any EJB session or message-driven bean.

For more information about interceptors, see "Using Interceptors in CDI Applications" in the *Java EE 6 Tutorial*.

To create an interceptor binding type:

1. Select a project in the Applications window and from the File menu, choose **New > From Gallery**.
2. From the New Gallery, expand **Business Tier** and **Contexts and Dependency Injection**, and then select **Interceptor Binding Type** and click **OK**.

3. In the Create Interceptor Binding Type dialog, enter a name and select the package where, and click **OK**.
4. In the Applications window, select the newly created file to open it in the Java source editor.

   The following example contains the annotations for a CDI interceptor binding type:

```
package demo;

import static java.lang.annotation.ElementType.METHOD;
import static java.lang.annotation.ElementType.TYPE;
import java.lang.annotation.Inherited;
import java.lang.annotation.Retention;
import static java.lang.annotation.RetentionPolicy.RUNTIME;
import java.lang.annotation.Target;

import javax.interceptor.InterceptorBinding;

@Retention(RUNTIME)
@Target({ METHOD, TYPE })
@InterceptorBinding
@Inherited
public @interface InterceptorBinding1 {
}
```

Qualifier Type

A qualifier type is a Java annotation that can be used to provide various implementations of a particular bean type.

For more information about qualifiers, see "Using Qualifiers" in the *Java EE 6 Tutorial*.

To create a qualifier type:

1. Select a project in the Applications window and from the File menu, choose **New > From Gallery**.
2. From the New Gallery, expand **Business Tier** and **Contexts and Dependency Injection**, and then select **Qualifier Type** and click **OK**.
3. In the Create Qualifier Type dialog, enter a name for the qualifier type and click **OK** to create it in the current package. You can also click **Browse** to select another package.
4. In the Applications window, select the newly created qualifier type Java file to open it in the Java source editor.

   The following example contains the annotation for a CDI qualifier type:

```
package demo;
import static java.lang.annotation.ElementType.FIELD;
import static java.lang.annotation.ElementType.METHOD;
import static java.lang.annotation.ElementType.PARAMETER;
import static java.lang.annotation.ElementType.TYPE;
import java.lang.annotation.Retention;
import static java.lang.annotation.RetentionPolicy.RUNTIME;
import java.lang.annotation.Target;

import javax.inject.Qualifier;

@Retention(RUNTIME)
@Target({ METHOD, FIELD, PARAMETER, TYPE })
@Qualifier
```

```
public @interface Qualifier1 {
}
```

Scope Type

For a web application to use a bean that injects another bean class, the bean needs to be able to hold state over the duration of the user's interaction with the application. The way to define this state is to give the bean a scope, such as `@RequestScoped`, `@SessionScoped`, or `@ApplicationScoped`.

For more information about scopes, see "Using Scopes" in the *Java EE 6 Tutorial*.

To create a scope type:

1. Select a project in the Applications window and from the File menu, choose **New > From Gallery**.
2. From the New Gallery, expand **Business Tier** and **Contexts and Dependency Injection**, and then select **Scope Type** and click **OK**.
3. In the Scope Type dialog, enter a name for the scope type and click **OK** to create it in the current package. You can also click **Browse** to select another package.
4. In the Applications window, select the newly created file to open it in the Java source editor.

   The following example contains the annotations for a CDI scope type:

```
package demo;
import static java.lang.annotation.ElementType.FIELD;
import static java.lang.annotation.ElementType.METHOD;
import static java.lang.annotation.ElementType.TYPE;
import java.lang.annotation.Inherited;
import java.lang.annotation.Retention;
import static java.lang.annotation.RetentionPolicy.RUNTIME;
import java.lang.annotation.Target;

import javax.inject.Scope;

@Retention(RUNTIME)
@Target({ METHOD, FIELD, TYPE })
@Scope
@Inherited
public @interface Scope1 {
}
```

Stereotype

A stereotype is a type of annotation, applied to a bean, that incorporates other annotations. Stereotypes can be particularly useful in large applications where you have a number of beans that perform similar functions.

For more information about stereotypes, see "Using Stereotypes in CDI Applications" in the *Java EE 6 Tutorial*.

To create a stereotype:

1. Select a project in the Applications window and from the File menu, choose **New > From Gallery**.
2. From the New Gallery, expand **Business Tier** and **Contexts and Dependency Injection**, and then select **Stereotype** and click **OK**.

3. In the Create Stereotype dialog, enter a name and select the package where, and click **OK**.
4. In the Applications window, select the newly created file to open it in the Java source editor.

   The following example contains the annotations for a CDI stereotype:

```
package demo;

import static java.lang.annotation.ElementType.FIELD;
import static java.lang.annotation.ElementType.METHOD;
import static java.lang.annotation.ElementType.TYPE;
import java.lang.annotation.Retention;
import static java.lang.annotation.RetentionPolicy.RUNTIME;
import java.lang.annotation.Target;

import javax.enterprise.inject.Stereotype;

@Retention(RUNTIME)
@Target({ METHOD, FIELD, TYPE })
@Stereotype
public @interface Stereotype1 {
}
```

Building a Persistence Tier

Learn how to build JPA entities to persist data objects in your application.

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.x is defined in the Java Persistence API.

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

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.

The following example contains the 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.

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 the example below:

```
@ManyToOne(cascade=PERSIST)
@JoinColumn(name="MANAGER_ID", referencedColumnName="EMP_ID")
public Employee getManager()
{
  return manager;
}
```

Inheritance and Polymorphism Support

Inheritance is very useful in many scenarios. The two types of inheritance that are commonly used and supported by for JPA entities are:

- Single table per class hierarchy
- Joined sub class strategy

The inheritance can be expressed using annotations. The following example contains code that uses the joined sub class 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
{
...
}
```

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 below.

```
@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);
}
}
```

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 Annotations for EJB/JPA.

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 Entities from Tables wizard allows you to create entities from online or emulated offline databases, as well as from an application server data sources.

> **Note:**
>
> When running the Entities from Tables wizard, if you configure the Attach an ID generator to each generated entity... field on the General Options page, you will need to manually configure the ID generator for the @SequenceGenerator or the @SequenceGenerator option, depending on which one you select. For additional instructions, refer to the Javadoc generated into each entity class.

To create entities or entities from tables:

1. From the main menu, choose **File > New**.
2. In the New Gallery, expand **Business Tier**, select **EJB** and then select **Entity** or **Entities from Tables** and click **OK**.

   Tip: Frequently-used selections are automatically saved to the File menu for easy access.
3. Follow the steps in the Create Entities from Tables or Create Entity wizard to create JPA entities.

   For more information at any time, press **F1** or click **Help** from within the wizard.

To create EJBs in an existing project:

1. In the Applications window, select the project in which you want to create a JPA entity and choose **New**.
2. In the New Gallery, expand **Business Tier**, select **EJB** and then select **Entity** or **Entities from Tables** and click **OK**.

   Or, from the main menu, choose **File > New** to open the New Gallery, and then follow step 2.

   Tip: Frequently-used selections are automatically saved to the File menu for easy access.
3. Follow the steps in the Create Entities from Tables or Create Entity wizard.

To create EJBs in a new project:

1. From the main menu, choose **File > New > Projects**.
2. In the New Gallery, choose the type of project you want to create and click **OK**.
3. In the Applications window, right-click the new project and choose **New** (or select the project and on the main menu choose **File > New**).

   In the New Gallery, expand **Business Tier**, select **EJB** and then select **Entity** or **Entities from Tables** and click **OK**.

   Tip: Frequently-used selections are automatically saved to the File menu for easy access.

Using the Serializable Interface

When creating an EJB using the **Entities from Tables** wizard, you can have the entity implement `java.io.Serializable` (on the General Options page of the wizard). When this is selected, a default `serialVersionUID` field is generated into the entity class, for example:

```
public class Departments implements Serializable {
    private static final long serialVersionUID = -1771169464233198257L;
...
```

Each time JDeveloper audits the class, it calculates the `serialVersionUID` that the VM would derive for the class, and compares it to the current `serialVersionUID` class property. If they are the same, then the class either has not changed, or has not changed in any meaningful way that affects serialization.

If the new `serialVersionUID` differs from the current `serialVersionUID` field, then JDeveloper flags it with an audit warning, and you must determine what to do about it.

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. SeeIntegrating Service-Enabled Application Modules.

The SDO architecture supported by JDeveloper offers the following:

- Simplifies the Java EE 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 Java EE 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 beans.

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 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://www.oasis-opencsa.org/sdo`

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 Java EE 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 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 deselect 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**.

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 the example belo] to the `weblogic-application.xml` which by default is located in `<workspace-directory>/src/META-INF`.

```
<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`.

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.

How to Generate Database Tables from JPA Entities

When you deploy a JPA entity to 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 How to Create JPA Entities.
2. Modify the entities as necessary, adding fields and constraints.
3. Name the tables:
   - EJB 3.x - 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 Deploying EJB Modules and JPA Persistence Units .

Annotations for EJB/JPA

Annotations simplify your development tasks by reducing the number of deployment descriptors needed for your application components. Annotations are also 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.x, see `http://download.oracle.com/javase/1.5.0/docs/guide/language/annotations.html`

> **Note:**
>
> Annotations are new to EJB 3.x, and not available for previous versions of EJB.

During design time, JDeveloper displays a list of available annotations through the Properties window. You can change any suitable Java class to an EJB or JPA component using the annotation feature. For more information, see How to Annotate Java Classes.

EJB 3.x

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 Singleton Session Bean. Choose **TRUE** or **FALSE** to annotate your class as a singleton session bean.
- Is Message Driven Bean. Choose **TRUE** or **FALSE** to annotate your class as a message driven bean.

JPA 2.0

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 Properties window displays a different set of contextual options, which you can use to add or edit annotations for the various members within the component class.

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 Properties window.

> **Note:**
>
> Annotations are only available for EJB 3.x, and not available for previous versions of EJB.

To annotate your Java class as an EJB/JPA component:

1. In the Applications window, 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 Properties window, 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 Properties window 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 Properties window.

> **Note:**
>
> EJB or JPA components created through the wizards already contain the class type annotations. For more information, see Building EJB 3.x Applications and Development Process..

Once your Java class is transformed into an EJB/JPA component using a class-level annotation, use the Properties window 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 Applications window, 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 Properties window, choose the tab corresponding to your EJB/JPA type.
4. Choose from any of the annotations available for the specific member you have selected.

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 Modeling with EJB Diagrams .

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`.

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 the example below:

```
@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 https://www.oracle.com/middleware/technologies/toplink-jpa-annotations.html.

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.x, 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 can make the Java service class able to run by generating a sample POJO Java client with a `main()` method that will display the running output in the Message pane, or you can generate a servlet-based client that will display the results in a well-formatted table in your browser.

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, from the main menu, select **File > New**, then in the New Gallery, expand **Business Tier** and select **EJB**, and then **Java Service Facade** and click **OK**.

You can also create a data control from a service facade. In the Applications window, 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.

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 the mapping descriptor XML (`orm.xml` file).

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 the example below:

```
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 entity mappings using a mapping descriptor (orm.xml):

Create a mapping file (orm.xml) for the persistence unit.

1. Open the `persistence.xml` file in the Overview editor.
2. Select the **General** tab, expand the JPA Mapping Descriptors section, and click **Create New JPA Mapping Descriptor**.
3. Open the `orm.xml` file in the Overview editor.
4. To add the desired entity, select the **General** tab, then select **Mapped Classes**, then **Entities**, and then click the **+** button.

   The orm.xml file is created and an entry is added for a single entity. When adding mappings to that entity, you will now be prompted to save those mappings using either annotations or XML (or both).

Configuring ID mapping in an orm.xml file

1. In the Overview editor, double click the `orm.xml` file you created in To configure entity mappings using a mapping descriptor (orm.xml):.
2. In the Structure pane, expand it until you have reached your entity. Then right-click the desired ID field and choose **Map As -> ID**. When prompted, choose **XML** to persist the metadata in the `orm.xml` file, which should look similar to the one shown in the example below:

```
<?xml version="1.0" encoding="windows-1252" ?>
<entity-mappings xmlns="<a target="_blank" href="http://www.eclipse.org/eclipselink/
xsds/persistence/orm"">http://www.eclipse.org/eclipselink/xsds/persistence/orm"</a>
                 xmlns:xsi="<a target="_blank" href="http://www.w3.org/2001/
XMLSchema-instance"">http://www.w3.org/2001/XMLSchema-instance"</a>
                 xsi:schemaLocation="<a target="_blank" href="http://www.eclipse.org/
eclipselink/xsds/persistence/orm">http://www.eclipse.org/eclipselink/xsds/
persistence/orm</a> <a target="_blank"                  href="http://www.eclipse.org/
eclipselink/xsds/eclipselink_orm_2_4.xsd"">http://www.eclipse.org/eclipselink/xsds/
eclipselink_orm_2_4.xsd"</a>
                 version="2.4">
  <entity class="model1.Departments">
    <attributes>
      <id name="departmentId"/>
    </attributes>
  </entity>
</entity-mappings>
```

Implementing Business Processes in Session Beans

Learn how to implement session beans which act to shield the client from complexity by executing business tasks inside the server.

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 How to Create a Session Bean.

There are three 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.
- **Singleton**. Singleton session beans offer similar functionality to stateless session beans but differ from them in that there is only one singleton session bean instance per application, as opposed to a pool of stateless session beans, any of which may respond to a client request. Like stateless session beans, singleton session beans can implement web service endpoints. With singletons, you can easily share state between multiple instances of an enterprise bean component or between multiple enterprise bean components in the application. Singleton session beans maintain their state between client invocations but are not required to maintain their state across server crashes or shutdowns.

Using Session Facades

With JDeveloper you can select to automatically generate your session facade methods any time you create a session bean through the Create Session Bean wizard. This creates a session bean that functions as a session facade for your business workflow. For more information, see 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 business objects and provides a higher level abstraction to the client.

Session facades can be stateful, stateless, or singleton, 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.

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. From the main menu, choose **File > New**.
2. In the New Gallery, expand **Business Tier**, select **EJB** and then select **Session Bean** and click **OK**.

   Tip: Frequently-used selections are automatically saved to the File menu for easy access.

   > **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**.

3. To make the bean a session facade select **Generate Session Facade Methods** on the EJB Name and Options page.

   For more information at any time, press **F1** or click **Help** from within the wizard.
4. Complete the remaining steps in the Create Session Bean wizard.

To add or remove session bean facade methods:

1. In the Applications window, select the session bean you want to edit.
2. Right-click and choose **Edit Session Facade**.
3. In the Specify Session Facade Options dialog, select a method on the list to expose it through the facade, or deselect a method so it will not be exposed.

For more information on session facades, see *Core J2EE Pattern - Sessions Facade* 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.x, 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 Applications window, 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. From the Applications window, open your EJB diagram.

   If you do not have an EJB diagram, from the main menu, select **File > New**. In the New Gallery, expand **Business Tier**, select **EJB Diagram** and then click **OK**.

2. In the Components window, click **Session Bean**.

   If the Components window is not visible, from the **View** menu, choose **Components**.

3. Click inside the EJB diagram (note that you do not drag and drop).

How to Create Session Beans in EJB Modules

You can create session beans in both new and existing EJB modules.

To create session beans in an existing EJB module:

1. In the Applications window, right-click an EJB project and choose **New > Session Bean**.

   Or, select the project and on the main menu choose **File > New > From Gallery**. In the New Gallery, expand **Business Tier**, select **EJB**, and then select **Session Bean** and click **OK**.

2. Follow the steps in the Create Session Bean wizard.

To create session beans in a new EJB module:

1. In the Applications window, select **File > New > Project**.
2. In the New Gallery, select the type of project you want to create and click **OK**.
3. In the Applications window, right-click the new project and choose **New > From Gallery**. In the New Gallery, expand the category for **Business Tier**, select **EJB**, and then select **Session Bean** and click **OK**.

   Or, select the project, and choose **New > Session Bean**.

4. Follow the steps in the Create Session Bean wizard.

How to Create Message-Drive Beans in EJB Modules

You can create EJBs in both new and existing modules.

To create message-driven beans in an existing EJB module:

1. In the Applications window, right-click an EJB project and choose **New > Message-Driven Bean**.

   Or, select the project and on the main menu choose **File > New > From Gallery**. In the New Gallery, expand **Business Tier**, select **EJB**, and then select **Message-Driven Bean** and click **OK**.

2. Follow the steps in the Create Message-Driven Bean wizard.

To create message-driven beans in a new EJB module:

1. In the Applications window, select **File > New > Project**.
2. In the New Gallery, select the type of project you want to create and click **OK**.
3. In the Applications window, right-click the new project and choose **New > From Gallery**. In the New Gallery, expand the category for **Business Tier**, select **EJB**, and then select **Message-Driven Bean** and click **OK**.

   Or, select the project, and choose **New > Message-Driven Bean**.

4. Follow the steps in the Create Message-Driven Bean wizard.

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:

1. In the Applications window, 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:

1. In the Applications window, 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:

1. In the Applications window, 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**.

How to Add a Field to an EJB

You can add fields to EJBs on an EJB diagram or through the EJB Module Editor.

1. In the Applications window, 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**.

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 Applications window:

1. In the Applications window, 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.

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.x Java Community Process specifications and documentation, see `http://www.oracle.com/technetwork/java/docs-135218.html`.

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 Implementing Business Processes in Session Beans and 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 Developing Applications with JavaServer Faces.

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.x resource referencess, go to your source code to annotate resource references.

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.x), 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. Create an ADF Data Control to create the XML definitions for each entity. For more information, see "Using ADF Data Controls" in *Developing Applications with Oracle ADF Data Controls*.
2. In the Applications window, select an EJB entity XML file.
3. In the Structure pane, select an entity attribute and then from the **View** menu, choose **Properties**.
4. In the Properties window, find the attribute you want as the primary key and set the **PrimaryKey** value to **true**.

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 more information, see "Using ADF Data Controls" in *Developing Applications with Oracle ADF Data Controls*.

Modeling EJB/JPA Components on a Diagram

You can model EJB and JPA components on a diagram.

See Modeling with EJB Diagrams for information about modeling these components in JDeveloper.

Deploying EJBs as Part of an Web Application

EJB classes can be packed inside archives for deployment.

EJB 3.1 has removed the restriction that enterprise bean classes must be packaged in an ejb-jar file. Therefore, EJB classes can be packaged directly inside a Web application archive (WAR) using the same packaging guidelines that apply to Web application classes. Simply put your EJB classes in the `WEB-INF/classes` directory or in a JAR file within `WEB-INF/lib` directory. Optionally, if you are also using the EJB deployment descriptor, you can package it as `WEB-INF/ejb-jar.xml`.

Deploying EJB Modules and JPA Persistence Units

Learn about deploying JPA entity beans and EJB modules.

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.

Deploying JPA Entity Beans

Entity beans were once only packaged in the EJB JAR file along with the session and message-driven beans. However, with JPA entities and the persistence unit technology, at deploy-time, they are packaged in their own JAR file, `persistenceunit.jar`.

This way 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.

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 About Deploying Applications.

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.x application that uses entities must define at least one persistence unit root either explicitly or using the JDeveloper default persistence unit. When you deploy your persistence unit, a JAR file is created called `persistenceunit.jar`. For more information, see About Deploying Applications.

How to Create a JPA Persistence Unit

You can create a persistence unit for your entities using the 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 Applications window and from the File menu, choose **New > From Gallery**.
2. From the New Gallery, expand **Business Tier** and **EJB**, and then select **JPA Persistence Unit** and click **OK**.
3. Complete the steps in the New Persistence Unit wizard.

How to Remove EJBs in a Module

To remove an EJB from an EJB module, select the EJB in the Applications window and press **Delete**.

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 Import 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 Import 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 Import wizard.

   To avoid conflicts, if an EJB with the same name already exists in your existing module, that EJB will not be imported.

Running and Testing EJB/JPA Components

Learn how to test EJBs in JDeveloper.

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.

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 session bean client on the integrated Oracle WebLogic Server:

1. In the Applications window, right-click a session bean and choose **Run**.

   Note in the Message pane that Oracle WebLogic Server has been launched.
2. Right-click a session bean and choose **Session Bean Client** from the context menu.
3. On the Create Sample Client dialog, specify whether you want to create a **Servlet Client** or a **Java Client.**
4. 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 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.
5. After your bean has been successfully started from the Applications window, right-click the sample client and choose **Run**.

   For Java clients, the Message pane shows you the running output log. For servlet clients, the results are shown in a well-formatted table in your browser.

To run a sample MDB client on the integrated Oracle WebLogic Server:

Before you can successfully run a sample MDB client on the integrated Oracle WebLogic Server, you must first create a corresponding JMS queue resource in the WebLogic Server domain.

1. Follow the JMS "Queue and Topic Destination Configuration" instructions in *Queue and Topic Destination Configuration*.
2. Use the following guidelines when creating your JMS module resources:
   a. Create a new test JMS module (for example, "TestJmsModule") with a new queue resource (for example, "DefaultQueue"), and
   b. Use a JNDI name that matches your generated MDBs Mapped Name, such as "weblogic.wsee.DefaultQueue".
   c. Add a default subdeployment resource, (for example, "DefaultQueue") and target it to the JMS server that is associated with the domain's *DefaultServer* instance.
3. In Applications window, select your MDB to create and run the sample client.

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 EJB versions in the same module.

To run a sample client on a remote server:

1. If necessary, create a connection to a running application server. For detailed instructions, see How to Create a Connection to the Target Application Server.
2. Create a project-level EJB JAR deployment profile:
   a. In the Applications window, right-click your project node and choose **Deploy > New Deployment Profile**.
   b. In the Create Deployment Profile dialog, choose a profile type of EJB JAR file and enter a name for the profile. When you click **OK** the Edit EJB JAR Deployment Profile Properties dialog opens. Accept the defaults and click **OK**.
3. Create an application-level, EAR-type deployment profile:
   a. Choose **Application > Deploy > New Deployment Profile**.
   b. In the Create Deployment Profile dialog, choose a profile type of EAR File and enter a name for the profile. When you click **OK** the Edit EAR Deployment Profile Properties dialog opens.
4. Add the new EJB JAR profile to the EAR profile file:
   a. In the Edit EAR Deployment Profile Properties dialog, choose **Application Assembly** in the navigation pane.
   b. Expand the Java EE Modules tree, and select the EJB JAR profile you created in Step 2 and click **OK**.
5. Deploy the application to the application server connection. Choose **Application > Deploy *application-deployment-profile***.
6. In the [Deploy] dialog box, choose **Deploy to Application Server** and click **Next**.
7. On the Select Server page, choose the application server connection and click **Finish**. You can track the deployment in the Deployment Log window
8. In the Applications window, right-click a session bean and choose **Session Bean Client**.
9. On the Create Sample Client dialog, specify whether you want to create a **Servlet Client** or a **Java Client.**
10. 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.
11. In the Applications window, right-click the new client and choose **Run**.

    For Java clients, the Message pane shows you the running output log. For servlet clients, the results are shown in a well-formatted table in your browser.

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. Right-click the EJB session bean or an ordinary Java class (POJO) in the Applications window (or you can navigate to it from within the wizard) and choose **New > From Gallery**.
2. From the New Gallery, expand **Business Tier** and **EJB**, and then select **EJB JUnit TestCase** and click **OK**.
3. Start the JUnit wizard.
4. Complete the steps in the wizard.

18

Developing Persistence in Applications Using Oracle TopLink

This chapter describes how to use the visual tools in JDeveloper to implement persistence using Oracle TopLink. You can configure TopLink descriptors and mappings for Java classes, EJBs, and JPA entities to data source elements (such as database tables or XML schema elements).
This chapter includes the following sections which describe the general process for creating TopLink mappings and integrating them in a JDeveloper project:

- About Developing Persistence in Applications Using TopLink
- Developing TopLink JPA Projects
- Developing Native TopLink Mappings
- Developing Native TopLink Relational Projects
- Developing Native TopLink XML Projects
- Developing Native TopLink EIS Projects
- Developing Native TopLink Sessions
- Developing Native TopLink Applications

Additional information is available at:

- Getting Started with Developing Java EE Applications
- Developing Applications Using Web Page Tools
- Developing with EJB and JPA Components
- *Understanding Oracle TopLink*
- *Java Persistence API (JPA) Extensions Reference for Oracle TopLink*
- *Developing Persistence Architectures Using Oracle TopLink Database Web Services*
- *Developing Persistence Architectures Using Oracle Toplink Document Data Bindings*
- *Java API Reference for Oracle TopLink*

About Developing Persistence in Applications Using TopLink

Learn about using TopLink with JDeveloper.

Oracle TopLink is an object-persistence and object-transformation framework that provides development tools and run-time capabilities.

TopLink links object-oriented programs with data structures. TopLink transforms object-oriented data into either relational data or XML documents. Using TopLink, you can integrate persistence and object-transformation into your application.

Using the tools in JDeveloper, you can configure and map Java classes, EJBs, and JPA entities to different data sources, including relational databases, enterprise information systems

(EIS), XML schemas, and JSON documents. TopLink supports multiple standards, including JPA, JAXB, and Java EE.

Oracle TopLink provides a complete, JPA 1.0 and 2.0-compliant JPA implementation. It provides complete compliance for all of the mandatory features, many of the optional features, and some additional features.

Developing TopLink JPA Projects

Use JDeveloper's TopLink tools to develop JPA projects.

The Java Persistence API (JPA) is a lightweight framework for Java persistence based on Plain Old Java Objects (POJOs). JPA is part of the EJB 3.x 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.

Oracle TopLink provides a complete, JPA 1.0 and 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.x container or outside the container. This includes Web containers, other non-EJB 3.x 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.x container.

The TopLink implementation of JPA is provided by EclipseLink. For more information, see `http://wiki.eclipse.org/EclipseLink`.

How to Specify the JPA Version

You can specify which version of JPA (1.0 or 2.0) to use at the project level. If you create a new mappings file, a new persistence unit, or a new persistence descriptor (`persistence.xml`), and the JPA version has not been selected yet, you have the opportunity to select the version. If one of those items has already been created in the project, the version for the project is used.

To Specify the JPA Version:

1. In a project that has not yet been associated with a JPA version, create any of the following:
   - Entity. See How to Create Entities.
   - JPA mappings descriptor (`orm.xml`). See How to Create JPA Mapping Descriptors.
   - JPA persistence descriptor (`persistence.xml`. See How to Create and Configure a JPA Persistence Descriptor (persistence.xml).)
2. Select the JPA version:
   - On the Version page of the Create Entity wizard or the Create Entities from Tables wizard, select **JPA 1.0 (Java EE 5)** or **JPA 2.0 (Java EE 6)**.
   - In the New Persistence Unit dialog or the New JPA Persistence Descriptor dialog, under **JPA Version**, select **JPA 1.0** or **JPA 2.0**.

> **Note:**
>
> Java EE 7 is supported.

How to Create Entities

Starting with EJB 3.x, a JPA entity is a Plain Old Java Object (POJO) that represents persistent data stored in a relational database or other data store. Typically, an entity represents a table in a relational database, and each entity instance corresponds to a row in that table. In JDeveloper, you can create an entity by specifying all the pertinent information, including details about the table structure in the database. You can also create entities from existing database tables.

If no persistence configuration (`persistence.xml`) yet exists in the project, a new one is created.

When you create entities from tables, mappings are automatically created, which can later be modified.

To create an entity:

1. In the Applications window, right-click the project in which you want to create an entity and choose **New**.
2. In the New Gallery, expand **Business Tier**, select **TopLink/JPA** and then select **Entity**, and click **OK**.
3. Complete the fields in the Create Entity wizard.

To create entities from existing database tables:

1. In the Applications window, right-click the project in which you want to create entities and choose **New**.
2. In the New Gallery, expand **Business Tier**, select **TopLink/JPA** and then select **Entities From Tables**, and click **OK**.
3. Complete the fields in the Create Entities From Tables wizard.

How to Create and Configure a JPA Persistence Descriptor (persistence.xml)

Use the persistence configuration file (`persistence.xml`) to configure the persistence context.

To create a persistence configuration:

1. In the Applications window, right-click the project in which you want to create a JPA persistence descriptor and choose **New**.
2. In the New Gallery, expand **Business Tier**, select **TopLink/JPA** and then select **JPA Persistence Descriptor**, and click **OK**.
3. Complete the fields in New JPA Persistence Descriptor dialog to create a default persistence unit for the new JPA persistence descriptor file (`persistence.xml`) and click **OK**.

   The following example contains a sample persistence configuration:

```
<?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="myPersistenceUnitName">
...
 </persistence-unit>
</persistence>
```

To configure a persistence configuration (persistence.xml) file:

1. In the Applications window, select the JPA Persistence descriptor (`persistence.xml`).
2. In the Structure window, select **JPA Persistence Descriptor**.
3. On the JPA Persistence descriptor (`persistence.xml`) page, complete the **General** and **Metadata Preferences** tabs.

How to Create Persistence Units

To create a persistence unit:

1. In the Applications window or Structure window, double-click the persistence descriptor (`persistence.xml`).
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 How to Create JPA Mapping Descriptors.)

   The following example contains a sample 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.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>
...
```

How to Configure Persistence Units

The tabs of the Persistence Unit page (accessed by first selecting `persistence.xml` in the Applications window 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. In the Applications window, double-click the JPA persistence descriptor (`persistence.xml`).
2. In the overview editor for `persistence.xml`, click the **General** navigation tab.
3. Under **Persistence Units**, double-click the name of the persistence unit you want to configure.
4. In the overview editor for the persistence unit, click the **General** navigation tab and then complete the fields to specify how the persistence unit connects to the application server and database.

To configure the connection information for a JPA persistence unit:

1. In the Applications window, double-click the JPA persistence descriptor (`persistence.xml`).
2. In the overview editor for `persistence.xml`, click the **General** navigation tab.
3. Under **Persistence Units**, double-click the name of the persistence unit you want to configure.
4. In the overview editor for the persistence unit, click the **Connection** navigation tab and then complete the fields 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. In the Applications window, double-click the JPA persistence descriptor (`persistence.xml`).
2. In the overview editor for `persistence.xml`, click the **General** navigation tab.
3. Under **Persistence Units**, double-click the name of the persistence unit you want to configure.
4. In the overview editor for the persistence unit, click the **Session Customization** navigation tab and then complete the fields to specify TopLink-specific information for the persistence unit. You can specify entity customizer classes, set TopLink session overrides, and customize entity manager properties, for example, validation mode, pessimistic locking mode, and query timeout mode.

To configure the DDL generation options:

Although most JPA persistence providers provide this support, these options are TopLink-specific.

1. In the Applications window, double-click the JPA persistence descriptor (`persistence.xml`).
2. In the overview editor for `persistence.xml`, click the **General** navigation tab.
3. Under **Persistence Units**, double-click the name of the persistence unit you want to configure.
4. In the overview editor for the persistence unit, click the **Schema Generation** navigation tab and then complete the fields to specify how the TopLink generates the DDL scripts.

To configure the non-TopLink specific properties for the persistence unit:

1. In the Applications window, double-click the JPA persistence descriptor (`persistence.xml`).
2. In the overview editor for `persistence.xml`, click the **General** navigation tab.
3. Under **Persistence Units**, double-click the name of the persistence unit you want to configure.
4. In the overview editor for persistence unit, click the **Properties** navigation tab and then complete the fields to specify the general, non-TopLink specific properties.

To configure metadata overrides for a persistence unit:

1. In the Applications window, double-click the JPA persistence descriptor (`persistence.xml`).
2. In the overview editor for `persistence.xml`, click the **General** navigation tab.
3. Under **Persistence Units**, double-click the name of the persistence unit you want to configure.
4. In the overview editor for the persistence unit, click the **Metadata Preferences** navigation tab and then complete the fields to specify the information for the mapping descriptor.

   This tab is available only if the persistence unit contains a JPA mapping descriptor.

About Using JPA Mappings

TopLink JPA supports the following types of mappings for an entity:

- **Basic** - A basic mapping defines a direct association between an entity field/property and a column on the database.
- **Element collection** - An element collection mapping maps an association collection of basic values or embeddables to the database.
- **Embedded** - An embedded mapping is used to specify a persistence property of an entity whose value is an instance of an embeddable class.
- **Embedded ID** - An embedded ID mapping designates a persistent field or property of an entity or mapped superclass which is a composite primary key that is an embeddable class. The referenced embeddable class must be an Embeddable.
- **ID** - An ID mapping designates the primary key property or field of an entity and may be applied in an entity or mapped superclass.
- **Many-to-many** - By default, JPA automatically defines a many-to-many mapping for a many-valued association with many-to-many multiplicity.

- **Many-to-one** - By default, JPA defines a many-to-one mapping for a single-valued association to another entity that has many-to-one multiplicity.
- **One-to-many** - A many-to-one mapping defines a many-valued association with one-to-many multiplicity. Bidirectional and unidirectional mappings are supported.
- **One-to-one** - A one-to-one mapping defines a single-valued association to another entity that has one-to-one multiplicity. Bidirectional and unidirectional mappings are supported.
- **Transformation** - A transformation mapping is used to map an attribute to one or more database columns. A read transformer and multiple write transformers can be configured.
- **Transient** - By default, all fields of an entity are assumed to be persistent. Use a transient mapping to specify a field or property of an entity that is not persistent, for example a field or property that is used at run time but that is not part of the entity's state.
- **Variable one-to-one** - A variable one-to-one mapping is used to represent a pointer reference between a Java object and an implementer of an interface. This mapping is usually represented by a single pointer (stored in an instance variable) between the source and target objects.

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**.

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. A basic mapping defines a direct association between an entity field/property and a column on the database. By default, TopLink persistence provider automatically configures a basic mapping for simple types. However, you can use the @Basic annotation to override defaults.

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`

How to Use JPA Mappings

The following procedures describe how to accomplish various tasks when working with JPA mappings.

To configure mappings using annotations:

1. In the Applications window, select the persistence configuration (`persistence.xml`).
2. In the Structure view, open the nodes down to the level of the mappings. That is, expand the project, expand the mapped classes, then expand the entity.
3. Do either of the following:
   - Select the field or property (for example, `CountryID`). In the Overview editor for the mapping, edit the details of the mapping, as desired.
   - Right-click the field or property, point to `Map As`, then choose a new mapping. In the Overview editor for the mapping, edit the details of the mapping, as desired.
4. To view the Java source, right-click the mapped field or property in the Structure view, and click **Go to Source**.

To configure mappings using a mapping descriptor file (`orm.xml`):

1. In the Applications window, select the mapping descriptor (`orm.xml`).
2. In the Structure view, open the nodes down to the level of the mappings. That is, expand the project, expand the mapped classes, then expand the entity.
3. Do either of the following:
   - Select the field or property (for example, `CountryID`). In the Overview editor for the mapping, edit the details of the mapping, as desired.

- Right-click the field or property, point to `Map As`, then choose a new mapping. In the Overview editor for the mapping, edit the details of the mapping, as desired.

4. To view the XML source:
   a. Right-click the mapped field or property in the Structure view, and click **Go to Source**.
   b. If the Overview editor for the mapping is displayed, click the **Source** tab to see the XML source.

How to Create JPA Mapping Descriptors

The JPA mapping descriptor (typically named `orm.xml`) can be used as an alternative to annotations. Information in the JPA mapping descriptor overrides the Java annotations.

To create JPA mapping descriptors:

1. In the Applications window, right-click the project in which you want to create a JPA persistence descriptor and choose **New**.
2. In the New Gallery, expand **Business Tier**, select **TopLink/JPA** and then select **JPA Mapping (XML)**, and click **OK**.
3. Complete the fields in New JPA Mappings dialog and click **OK**.

To create JPA mapping descriptors associated with a persistence unit:

1. In the Applications window, double-click the persistence configuration (`persistence.xml`).
2. In the overview editor for `persistence.xml`, click the **General** navigation tab.
3. In the Persistence Units area, double-click the persistence unit.
4. Click the **General** tab for the persistence unit,
5. In the JPA Mapping Descriptors area, click the **Create New JPA Mapping Descriptor** button.
6. Complete the fields in the dialog and click **OK**.
7. In the JPA Mapping Descriptors area, double-click the name of the new mapping descriptor and then specify the details for the descriptor (such as mapped classes, development database, and other defaults).:

To configure the general information for a JPA mapping descriptor:

1. In the Applications window, double-click the JPA mapping descriptor (`orm.xml`).
2. In the overview editor for the mapping descriptor, click the **General** navigation tab, and then complete the fields to select a persistence provider and configure its general properties (such as mapped classes, development database, and other defaults).

To associate entities, embeddables, or mapped classes with a JPA mapping descriptor:

1. In the Applications window, double-click the JPA mapping descriptor (`orm.xml`).
2. in the Overview editor for the descriptor, click the **General** navigation tab.
3. In the Mapped Classes area, do any of the following:
   - Under **Entities**, click the **Add Classes as Entities** icon.
   - Under **Embeddables**, click the **Add Classes as Embeddable Classes** icon.
   - Under **Mapped Superclasses**, click the **Add Classes as Mapped Superclasses** icon.

4. In the Manage Entity Classes dialog, the Manage Embeddable Classes dialog, the Manage Mapped Superclasses Classes dialog, do the following:
   a. Select the project.
   b. Under **Available Classes**, expand the node and select the items to add.
   c. Click the **Add Selected Classes to List** or **Add All Classes to List** icon to move the selected items to the **Selected Classes** list. Then click **OK**.

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. In the Applications window, double-click the JPA mapping descriptor (`orm.xml`).
2. In the overview editor for the mapping descriptor, click the **Persistence Unit Defaults** navigation tab and then complete the fields to configure the access type, entity listeners, multitenancy, delimited identifiers, and other defaults.

How to Set Access Type Defaults and Overrides

You can specify whether the persistent state of managed class attributes is accessed on fields or properties. Field values are accessed directly, and property values are accessed using `get()` and `set()` methods.

You can set defaults or specify specific access types at various levels in the configuration. Using defaults and overrides is sometimes called "mixed access."

By default, a single access type (field or property) applies to an entity hierarchy, although the default can be overridden, as described in the following sections.

The order of precedence of access type settings is described in the following list, from highest precedence to lowest. For managed classes (entities, embeddables, and mapped superclasses), the setting for a class or an attribute lower in an inheritance hierarchy overrides the setting higher in the hierarchy.

1. Mapped attribute. A setting at this level overrides the default settings on managed classes, mapping descriptors (`orm.xml`), and persistence units.
2. Managed classes. A setting at this level overrides the default settings for mapping descriptors (`orm.xml`) and persistence units. It can be overridden by settings on mapped attributes.

   The access type of an embeddable class is determined by the access type of the entity class, mapped superclass, or embeddable class in which it is embedded (including as a member of an element collection) independent of whether the access type of the containing class has been explicitly specified or defaulted. You can override the inherited access type by specifying a different type for the embeddable class.
3. Mapping descriptors (`orm.xml`). A setting at this level overrides the default settings for the persistence unit. It can be overridden by settings on managed classes and mapped attributes.
4. Persistence units. A setting at this level can be overridden by settings for mapping descriptors, managed classes, and mapped attributes.

To configure access type for managed classes in the persistence unit:

1. In the Applications window, double-click the JPA mapping descriptor (`orm.xml`).
2. In the overview editor for the mapping descriptor, click the **Persistence Unit Defaults** navigation tab.
3. Under **Access Type**, select **Field Accessing** or **Method Accessing** to define the default for all the managed classes in the persistence unit that have XML entries in the mapping descriptor.*

To configure access type for managed classes in the mapping descriptor:

1. In the Applications window, double-click the JPA mapping descriptor (`orm.xml`).
2. In the overview editor for the mapping descriptor, click the **General** navigation tab.
3. In the **Defaults** area, under **Access Type**, select **Field** or **Property**.

To configure access type for a managed class (entity, embeddable class, or mapped superclass):

1. In the Applications window, double-click the Java source file for the managed class).
2. In the Overview editor for the class, click the **General** navigation tab.
3. Under **Access Type**, select **Field** or **Property**

To specify method accessing for attribute mappings:

1. In the Applications window, double-click the JPA mapping descriptor (`orm.xml`).
2. In the Structure window, expand the managed class containing the attribute mapping you want to configure, then double-click the attribute.
3. Select **Use Method Accessing**, then select the desired `set()` and/or `get()` method.

How to Generate Unique IDs for Primary Keys

Define generators to determine the generator strategy for an entity and the named generator be used to assign a unique ID to an entity. The scope of a generator name is global to the persistence unit across all generator types. A generator defined using XML in a mapping descriptor overrides a generator of the same name defined using an annotation on an entity.

To configure generators in the JPA mapping descriptor (orm.xml):

1. In the Applications window, double-click the JPA mapping descriptor (`orm.xml`).
2. In the overview editor for the mapping descriptor, click the **Generators** tab to configure sequence generators and table generators.

To configure generators for entities

1. In the Applications window, double-click the source file for the entity, for example `Departments.java`.
2. In the overview editor for the entity, click the **Generators** tab to configure sequence generators and table generators.

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. In the Applications window, double-click the JPA mapping descriptor (`orm.xml`).
2. In the overview editor for the mapping descriptor, click the **Queries** navigation tab to define named queries, named native queries, and named stored procedure queries.

How to Specify Derived Identifiers in Mappings

In one-to-one and many-to-one mappings, you can specify that the identity (Id) for an entity is derived from the Id of its parent entity (the target of the mapping).

To specify that an identifier Is derived:

1. In the Applications window, click the configuration file that specifies the mappings, `persistence.xml` or the JPA mapping descriptor (`orm.xml`).
2. In the Structure window, expand the class containing the one-to-on or many-to-one mapping for which you want to specify a derived identifier, then click the attribute.
3. Select **Derived Identity**, then select one of the following:
   - **None** - Do not derive an `Id`.
   - **Id** - Select if the `Id` is a single value, in which case the source object's `Id` is the same as the target object's `Id`.
   - **Maps Id** - Select an attribute to provide the mapping for an `EmbeddedId` primary key, an attribute within an `EmbeddedId` primary key, or a simple primary key of the parent entity.

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.

Developing Native TopLink Mappings

Use native TopLink with JDeveloper.

Using the tools in JDeveloper, you can use native TopLink to configure and map Java classes, EJBs, and JPA entities to different data sources, including relational databases, enterprise information systems (EIS), and XML schemas. TopLink supports multiple standards, including JPA, JAXB, and Java EE.

Designing Native TopLink Applications

Use JDeveloper to develop native TopLink applications.

You can use native 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 Oracle WebLogic Server or Apache Tomcat
- Java clients such as Java SE and web browsers

Using Native TopLink in Application Design

Native TopLink can be used in the following ways:

- **Relational Database Usage**: You can use native 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 native 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 native 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).

Creating Native TopLink Metadata

Native 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`, TopLink JPA annotation extensions, and TopLink property extensions. The TopLink JPA persistence provider interprets these sources of metadata to create an in-memory 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 or JPA entities) and the code written with the TopLink API, to complete the application.

Using native 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.

Creating Project Metadata

A native 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.

Creating Session Metadata

The native 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.

Using Native 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

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.

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.

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.

Using Native TopLink Mappings

Native 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.

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 18-1 illustrates the relational mapping types build maps using the TopLink concepts of directionality, transformers, converters, and EJB 2.n CMP relational mapping.

Table 18-1 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.

EIS Mapping Types

Native 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 18-2 illustrates the EIS mapping types that TopLink provides:

Table 18-2 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.

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 18-3 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 18-3 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.

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 3.1 JPA.

The TopLink editor displays the information or properties specific to the element selected in the Applications window or the Structure view. For example, selecting TopLink project elements in the Applications window, 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.

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 Applications window 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.

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.

Managing TopLink Sessions

The TopLink Sessions configuration file (sessions.xml) enables you to manage all of the sessions for a specific project. See *Understanding 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.

Managing Persistence Configurations

The TopLink editor enables you to configure the `persistence.xml` file, which packages entities in TopLink JPA projects. By choosing Applications window 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.

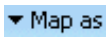
The TopLink Structure View Toolbar

The Structure view displays detailed information about the TopLink element selected in the Applications window 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 18-4 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.

TopLink Project Elements in the Applications Window

The Applications window displays each element associated with your TopLink project, including the TopLink Map, deployment descriptors, and sessions configuration information.

TopLink project elements in the Applications window may include:

- TopLink folder
- Sessions configuration file (`sessions.xml`)
- TopLink map (tlMap)

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 Applications window or Structure view.

TopLink Project Elements in the Structure View

The Structure view displays detailed information about the TopLink element selected in the Applications window 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 Applications window 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.

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.

TopLink Mapping Status Report in Message Log

Error and status messages from the TopLink Editor appear in the Issues window.

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. From the main menu, choose **Tools** > **Preferences**.
2. In the **Categories** list, expand **TopLink Customization**.
3. Configure JPA and Query options.
4. Complete each field and click **OK**.

How to Create a Native TopLink Mapping Project

JDeveloper stores the native TopLink descriptors (see Using Descriptors) and mappings (see Using 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 add a Native TopLink Object Map to an existing project:

1. In the Applications window, right-click the project to which you want to add a TopLink map and choose **New**.
2. In the New Gallery, expand **Business Tier**, select **TopLink/JPA** then select **TopLink Object Map**, and click **OK**.
3. Complete each field and click **OK**.

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.

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 Applications window, select a TopLink map.
2. In the Structure window, right-click the TopLink map (or a specific Java class or TopLink descriptor) and choose **Automap**.

3. Follow the steps in the Automap Wizard.

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.

Developing Native TopLink Relational Projects

Learn how to use TopLink to map Java objects to conventional database objects.

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.

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 native TopLink object map for a relational project:

1. If you are creating a new project for the relational project, create a new project, as follows:
   a. From the main menu, elect **File > New > Project**.
   b. In the New Gallery, select **Projects > Custom Project**.
2. In the Applications window, right-click the project in which you want to create a TopLink object map and choose **New > From Gallery**.
3. In the New Gallery, expand **Business Tier**, select **TopLink/JPA** and then select **TopLink Object Map**.
4. Click **OK**.
5. In the Data Source area of the New TopLink Object Map dialog, select **Database**, then specify your database information.
6. Click **OK**.

   The new project includes the TopLink map and a TopLink sessions configuration file (`sessions.xml`).

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 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. In the Applications window, right-click the TopLink map and then select **Add or Remove Descriptors**.
2. Select the packages and classes from which to create TopLink descriptors and click **OK**.

   The descriptors are added to the TopLink map in the Structure window.

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. In the Applications window, select the TopLink map.
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**.

Developing Native TopLink XML Projects

Use an XML project for non-transactional 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 compiler 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 compiler 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. See `https://www.oracle.com/java/technologies/jaxb.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.

How to Create XML Projects and Object Maps

To create a new native TopLink object map for an XML project:

1. If you are creating a new project for the XML project, create it as follows:
   a. From the main menu, elect **File > New > Project**.
   b. In the New Gallery, select **Projects > Custom Project**.
2. In the Applications window, right-click the project in which you want to create the TopLink XML object map and choose **New > From Gallery**.
3. In the New Gallery, expand **Business Tier**, select **TopLink/JPA** and then select **TopLink Object Map**.
4. Click **OK**.
5. In the Data Source area of the New TopLink Object Map dialog, select **XML**.
6. Click **OK**.

   JDeveloper adds the TopLink map and TopLink sessions configuration file (`sessions.xml`).

How to Create XML Descriptors

To create new TopLink descriptors for an XML project:

1. Right-click the TopLink map in the Applications window 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.

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. In the Applications window, select the TopLink map.
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.

Developing Native TopLink EIS Projects

Use TopLink EIS projects 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.

How to Create EIS Projects and Object Maps

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.

An EIS mapping transforms object data members to the EIS record format defined by the object's descriptor.

To create a new native TopLink object map for an EIS project:

1. If you are creating a new project for the EIS project, create it as follows:
   a. From the main menu, elect **File > New > Project**.
   b. In the New Gallery, select **Projects > Custom Project**.
2. In the Applications window, right-click the project in which you want to create a TopLink object map and choose **New > From Gallery**.
3. In the New Gallery, expand **Business Tier**, select **TopLink/JPA** and then select **TopLink Object Map**.
4. Click **OK**.
5. In the Data Source area of the New TopLink Object Map dialog, select **EIS**, then specify your EIS platform.
6. Click **OK**.

   JDeveloper adds the TopLink map and TopLink sessions configuration file (`sessions.xml`).

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.

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 Advanced Queuing (AQ)
- Attunity Connect
- IBM MQSeries
- Java Message Service (JMS)
- Blackbox

- XML file

This platform configuration is overridden by the session login, if configured.

Developing Native TopLink Sessions

Create native TopLink sessions which act as facades to access TopLink functionality.

Each TopLink map belongs to a TopLink 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. See the *Oracle® Fusion Middleware Developer's Guide for Oracle TopLink*.

How to Create a New Native TopLink Sessions Configuration File

Each native 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 the Applications window, 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.

How to Create Native TopLink Sessions

A native 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 Applications window, 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.

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.

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 Applications window, 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**.

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. In the Applications window, select the TopLink sessions configuration (`sessions.xml`).
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. If a login type has not yet been set for the session, select the type from the list:
   - **Database** - Select to configure connection information at the session level for a non-CMP TopLink application. The TopLink runtime uses this information whenever you perform a persistence operation using the session in your non-CMP TopLink application.
   - **EIS** - Select to configure connection information at the session level for an XML application. The TopLink runtime uses this information whenever you perform a persistence operation using the session in your EIS application.
   - **XML** - Use this page to specify the data source login settings for the TopLink XML session.

   For information about the options on the Login pages for Database, EIS, and XML, display the page (as described above) and press **F1** to consult the online help.

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. In the Applications window, select the TopLink sessions configuration (`sessions.xml`).
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.

Developing Native TopLink Applications

Learn how to create applications using native TopLink.

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 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.

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.

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.

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.

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.

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.

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.

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.

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.

How to Configure the TopLink Cache

The TopLink cache is an in-memory repository that stores recently read or written objects based on class and primary key values.

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. In the Applications window, select the TopLink map.
2. In the Structure window, double-click the TopLink map.
3. Select the **Defaults** tab.
4. Under **Caching**, set the caching options. You can select the following types of caching: **Full**, **None**, **Soft**, **Weak**, **Weak with Hard Subcache**, and **Weak with Soft Subcache**. For information on these caching types and their options, press **F1** and consult the online help.

To configure the TopLink cache at the descriptor level:

1. In the Applications window, select the TopLink map.
2. In the Structure window, expand the map to display its descriptors, then select the descriptor you want to configure.
3. Select the **Caching** tab.
4. Set the caching options. You can select the following types of caching: **Full**, **None**, **Soft**, **Weak**, **Weak with Hard Subcache**, and **Weak with Soft Subcache**. For information on these caching types and their options, press **F1** and consult the online help.

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.

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

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.

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. In the Applications window, select the TopLink map.
2. In the Structure window, select the descriptor.
3. On the **Queries** tab, create the query.

   The Queries tab allows you to create and manage queries associated with a TopLink descriptor. Available actions are:

   - **Named Queries**: Create named queries of types `ReadObjectQuery`, `ReadAllQuery`, and `ReportQuery`.
   - **Custom Calls**: Create custom queries.
   - **Query Keys**: Query keys are schema-independent aliases for database field names and are supported in relational database projects only.
   - **Settings**: Set query time-outs and cache refresh options.

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

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 information about advanced query API, see the *Oracle® Fusion Middleware Developer's Guide for Oracle TopLink*.

Redirect Queries

A redirect query is a named query that delegates query execution control to your application. Redirect queries 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.

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.

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.

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.

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.

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.

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

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.

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`.

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. In the Applications window, select the TopLink map.
2. In the Structure window, select the descriptor.
3. Select the named query and in the **Selection Criteria** area, edit the expression.

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.

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.

19

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:

- About Developing Secure Applications
- Securing Applications in Phases
- About Web Application Security and JDeveloper Support
- Handling User Authentication in Web Applications
- Securing Application Resources in Web Applications
- Configuring an Application-Level Policy Store
- Migrating the Policy Stores
- Securing Development with JDBC

About Developing Secure Applications

Learn how to develop 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).

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.

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.

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. See Introduction to Oracle Platform Security Services.

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.

See “Enabling ADF Security in a Fusion Web Application".

See Introduction to Oracle Platform Security Services.

Securing Applications in Phases

Learn how JDeveloper supports security during the development phase, and during deployment to the production environment.

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 need 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 Configuring LDAP Authentication Providers.

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

About Web Application Security and JDeveloper Support

Learn about security for web applications.

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 Introduction to Oracle Platform Security Services.

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. See 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. See. How to Secure ADF Resources Using ADF Security in Fusion Web Applications.

For information on web application security, see WebLogic Security Programming Overview.

Handling User Authentication in Web Applications

Authentication in declarative security is enforced when a user requests a protected web application area.

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.

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 below demonstrates how BASIC authentication is specified in the `web.xml` file:

```
<login-config>
<auth-method>BASIC</auth-method>
<realm-name>jazn.com</realm-name>
</login-config>
```

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 below demonstrates how FORM authentication is specified in the `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>
```

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 information about authentication type using Oracle WebLogic Server, see Developing Secure Web Applications. For information about users, enterprise roles, and realms, see the Terminology.

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.

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 19-1 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 19-1 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).

Table 19-1 (Cont.) Arguments for the weblogic.security.Encrypt utility

Argument	Definition
`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
```

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.

For information about the terms used in this section, see Terminology.

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.

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.

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 Applications window.
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.

How to Add Test Users to the Identity Store

The identity store is an XML file 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 Applications window.
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.

Managing Enterprise Roles in 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

How to Add Roles to the Identity Store

You can add roles to the identity store using the overview editor for the `jazn-data.xml` file.

To add roles to the identity store:

1. Open your application in the Applications window.
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.

How to Manage Users Assigned to User Roles

You can manage roles in the identity store using the overview editor for the `jazn-data.xml` file.

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.

How to View Assigned Enterprise Roles

You can view assigned roles in the identity store using the overview editor for the `jazn-data.xml` file.

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.

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 Applications window.

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 Applications window.
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.

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 19-2 contains the pre-defined login modules that are available in JDeveloper:

Table 19-2 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.WSSDigestLogin`Module. This is supported only for JSE use cases

Table 19-2 (Cont.) Predefined Login Modules

Module	Description
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.
`idstore.loginmodule`	Used to authenticate JSE bases use cases and implements `oracle.security.jps.internal.jaas.module.idstore.IdStoreLoginModule` class

See Using Login Modules in Java Applications.

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 Applications window.
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.

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 Applications window.
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.

How to Add a Key Store

A **keystore** 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 keystore in JDeveloper.

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 Applications window.
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 keystore 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.

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.

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 Applications window.
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.

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 Applications window.
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.

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. See About Authentication Type Choices.

To select the authentication type for the web application:

1. Double-click the `web.xml` for the application in the Applications window.
2. Click the **Security** tab of the `web.xml` Overview Editor.
3. Expand the **Login Authentication** section and select the desired authentication type.

Securing Application Resources in Web Applications

Learn how to secure web pages and other resources of the web application.

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*.

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 Applications window.
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.

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.

Configuring an Application-Level Policy Store

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.

A **Policy Store** is the repository of application and enterprise policies. A policy specifies the permissions granted to code running from a specific location.

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.

See Managing Policies.

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.

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 Applications window.
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.

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 Applications window.
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.

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 Applications window.
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.

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 Applications window.
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.

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 Applications window.
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.

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 Applications window.
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.

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 Applications window.
2. In the Applications window, 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.

Migrating the Policy Stores

JDeveloper's default behavior is to deploy security objects to Integrated WebLogic Server. Learn how to change this behavior.

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.

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**.

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 Migrating the Security Store with migrateSecurityStore.

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 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.

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.

Securing Development with JDBC

Use JDBC to secure database connections.

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 below:

```
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) ))
```

20

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:

- About Developing Applications Using XML
- Using the XML File Editors.
- Working with XML Schemas
- How to Import and Register XML Schemas
- How to Generate Java Classes from XML Schemas with JAXB
- Working with XSD Documents and Components
- Localizing with XML
- Developing Databound XML Pages with XSQL Servlet

About Developing Applications Using XML

Learn about the JDeveloper tools to work with 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 create your JSPX, XSD, XSQL, and configuration files using the wizards, and edit your files in the XML editor.

Using the XML File Editors

Learn about JDeveloper's XML editors.

There are three different editors for your XML files. Each editor has a specific function.

Table 20-1 XML File Editors

Editor	Description
XML Editor	A specialized schema-driven editor for editing XML languages, including `.xsql`, `.xsl`, `.xsd`, `.xhtml`, and `.wsdl` files. To open the editor, double-click a file type in the Applications window. The Source tab displays the source code for the file, which you can edit.
Overview Editor	View and edit XML (`.xml`) files. Visually displays aspects of your deployment-related XML files such as filters, security and references. See Developing Applications Using Web Page Tools.
XSD Visual Editor	Create or edit XML schemas. Visually displays the structure, content, and semantics of an XML document. See Working with XSD Documents and Components.

Understanding XML Editing Features

Table 20-2 summarizes the editing features that are available when you're working with XML files.

Table 20-2 XML Editing Features

Feature	Purpose
Code Insight	While you are typing, you can invoke Code Insight by pausing after typing < (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 Applications window, 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	You can customize these features on 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.
Components Window	Choose **Window** > **Components** to open the Components window 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 window.
Properties Window	The Properties window displays attributes of elements in the file. You can edit the values of attributes in the Properties window to update your file.
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.

Table 20-2 (Cont.) XML Editing Features

Feature	Purpose
Validate XML	In an open XML editor window, or in the Applications window, 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.

Understanding the XML Editor Toolbar

Table 20-3 contains the icons that display on the XML Editor toolbar.

Table 20-3 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.

How to Set Editing Options for the XML Editor

The XML Editor has the following editing features:

- **Required Attribute Insertion** - Required attributes that are associated with a tag are added automatically to your code, when you add the tag.
- **End Tag Completion** - Automatically adds end tags when you close a start tag. For example, in an XML file, if you type `<foo>`, `</foo>` is added automatically.

To customize editing options for the XML Editor:

1. Choose **Tools** > **Preferences**.
2. Expand the **Code Editor** node.
3. Select the **JSP/XML/HTML** node.
4. On the Code Editor - JSP/XML/HTML page, select an option.
5. Click **OK**.

Working with XML Schemas

Learn how to create and edit XML schemas.

XML schemas define the elements of your XML files. JDeveloper provides an XSD Visual Editor that gives a visual representation of the structure, content, and semantics of an XML document. Use the XSD Visual Editor to author a new XML schema (`.xsd` file) or to edit an existing XML schema.

You can insert components into the XSD document using the Components window or by right-clicking on a location in the XSD document.

The XML Schema component displays at the top of an XSD file, as shown in the figure. Right-click the element and select **Properties** to display a dialog for configuring the schema namespaces.

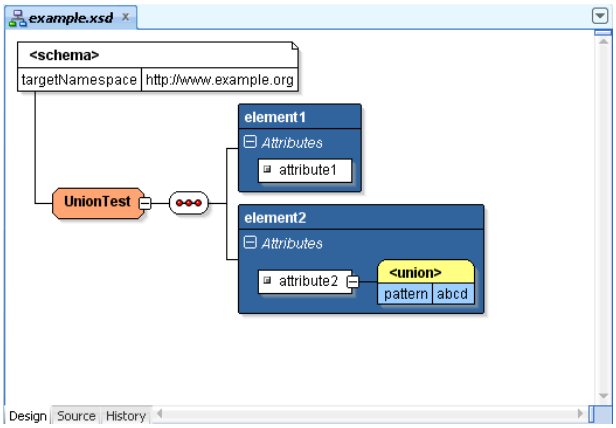
Figure 20-1 XML Schema Component

JDeveloper supports various refactoring operations on XML Schemas, such as changing the schema's target namespace and changing the base type on a simpletype element that has facets. For more information, see Refactoring XML Schemas.

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. The figure contains an example XML schema in the Design tab of the XSD Visual Editor.

Figure 20-2 Schema in XSD Visual Editor



You can edit attributes in `attribute2` in the attribute editor, which is displayed in the figure 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 subtler containing child nodes like list or union.

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. The following is the source for the `example.xsd` file shown in Figure 20-2.

```
<?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>
```

Selecting XSD Components

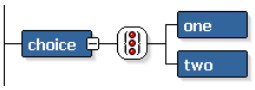
The selection of any component or attribute in the editor is indicated by highlighting the selected item in blue. In the figure, the selected `simpleType` component defines a simple type and specifies the constraints and information about the values of attributes or text-only components, in this case restricting the string type.

Figure 20-3 simpleType Component

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 the figure. Set attribute `maxOccurs` to >1 to have more than one item from the choice in the parent.

Figure 20-4 Choice Component

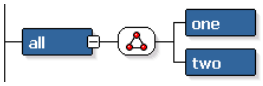



```
?xml version="1.0" encoding="windows-1252" ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns="http://www.example.org"
            targetNamespace="http://www.example.org" elementFormDefault="qualified">
  <xsd:element name="choice">
    <xsd:complexType>
      <xsd:choice>
        <xsd:element name="one"/>
        <xsd:element name="two"/>
      </xsd:choice>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

All Component

The all component shown in the figure specifies that the child components can appear in any order and that each child component can occur zero or one times.

Figure 20-5 All Component

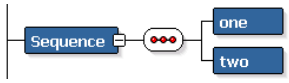



```
<?xml version="1.0" encoding="windows-1252" ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns="http://www.example.org"
            targetNamespace="http://www.example.org" elementFormDefault="qualified">
  <xsd:element name="all">
    <xsd:complexType>
      <xsd:all>
        <xsd:element name="one"/>
        <xsd:element name="two"/>
      </xsd:all>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Sequence Component

The sequence component shown in the figure specifies that the child components must appear in a sequence. Each child component can occur from 0 to any number of times.

Figure 20-6 Sequence Component




```
<?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:element name="all">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="one"/>
        <xsd:element name="two"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Cardinality and Ordinality

In the example of cardinality shown in the figure, components are displayed with the attributes shown in Table 20-4.

Figure 20-7 Cardinality Component

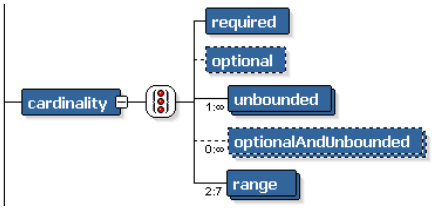


Table 20-4 Cardinality Display

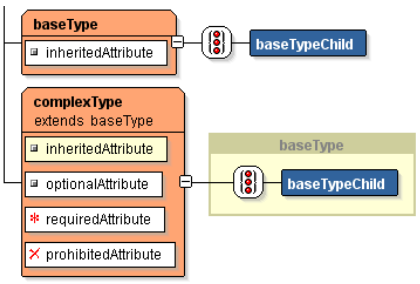
Component	Display
Required components (minOccurs=">0")	Display with a solid line.
Optional components (minOccurs="0")	Display with a dotted line.
Unbounded components (maxOccurs="unbounded")	Display an infinity symbol in the component stack number. Any component that can appear more than once is displayed 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). The number after the colon indicates the maximum number of times the component can occur (maxOccurs). In Figure 20-7, the maximum is unbounded so an infinity symbol is displayed.
Range of components	Display in the component stack number. In Figure 20-7, the component must appear at least 2 times in the instance document, but no more than 7.

ComplexType Component

In the figure, 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 20-8 complexType Component

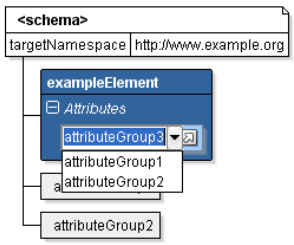


Attribute Group Component

The attribute group component groups a set of attribute declarations so that they can be incorporated as a group into complex type definition.

The figure displays three attribute groups.

Figure 20-9 Attribute Group Component

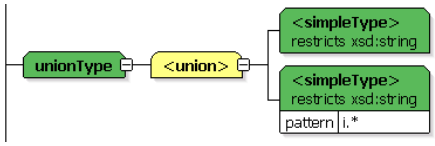


If you add an element to a schema that has multiple `attributeGroups`, you can add choose one or more `attributeGroups` for the element by clicking on the element's attribute and choosing from a drop-down list.

Union Component

The union component defines a simple type as a collection (union) of values from specified simple data types. In the figure, the union represents all strings that begin with the letter "i".

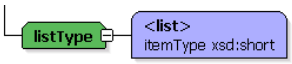
Figure 20-10 Union Component



List Component

The list component defines a simple type component as a space separated list of values of a specified data type. In the figure, the component represents a series of short value objects.

Figure 20-11 List Component



Working with XML Schema Substitution Groups

One element can substitute for another element in an XML Schema group. If you have a set of XML Schemas that use the substitution group feature, ensure that the XML Schema that defines the head of the substitution group is loaded before loading additional schemas that place elements into that substitution group.

For example, you might have an XML Schema named `A.xsd` that contains

```
<element name="A"/>
```

and another XML Schema named `B.xsd` that contains

```
<element name="B" substitutionGroup="namespaceForA:A" />
```

Ensure that one of the following is true:

- You call `SchemaGrammarProvider.add(A.xsd)` before calling `SchemaGrammarProvider.add(B.xsd)`. Or,
- `B.xsd` contains an import or include statement with a `schemaLocation` attribute that contains a valid path to `A.xsd`.

How to Import and Register XML Schemas

Learn how to import 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.

JDeveloper automatically validates the schema when you add or modify it.

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 Components window 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.

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** > **From Gallery** >**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.

Working with XSD Documents and Components

Use the XSD Visual Editor or Design structure window to work with XML Schema files (`.xsd` files) and components. By default, new schema files are opened with the XSD Visual Editor in focus.

Double-clicking a file in the Applications window 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

How to Display a Schema in Both Editors

Edit a schema file (`.xsd`) simultaneously in 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.

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.

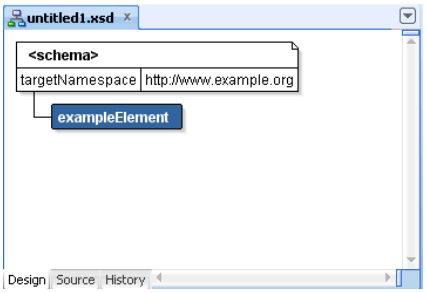
Supported image formats are `.svg`, `.svgz`, `.jpg`, and `.png`.

To save the XSD editor design tab as an image:

1. In the Applications window, 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 Developing Databound XML Pages with XSQL Servlet.

Figure 20-12 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.

> **Note:**
>
> If the diagram you are attempting to save is too large, a message displays to save 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.

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.

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.

- Click the + (plus) sign of the parent component to expand one level beyond the parent component.
- Click the - (minus) sign of the parent component to collapse all levels below the parent component.
- Press Ctrl + *, using the * on the numeric keypad of the keyboard to expand all parent components in the schema. Keep in mind that this view can be very large.

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. After placing your cursor in the area of the XML Schema you want to magnify

- Press Ctrl+Plus to zoom in. You can also use the Plus key on the numeric keypad of the keyboard

- Press Ctrl+Minus to zoom out. You can also use the Minus key on the numeric keypad of the keyboard.

How to Select XSD Components

One of the most common actions you perform in the editor is to select components. you will select components to:

- Edit properties
- Move components
- Delete components
- Choose a target position to insert another component

Escalating a Single Component

You can select a single component without children.

To select a single 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 the Properties window for the component.

Selecting Multiple Components

You can select a component along with its children, or multiple components.

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.

> **Note:**
>
> 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.

What Happens When You Select a Component in the XSD Visual Editor

When you select a component in the XSD Visual Editor, the component displays in blue. When a container component is selected and any of its children, all are blue.

When selected in the Structure window, the component is highlighted. However, when you select any components with children, the children are also selected. If you delete or move the parent, all the children are deleted or moved.

Whenever you select an component, you are also selecting a position in which another component can be inserted. For more information, see 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.
>
> 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. You can look at the selection in both tools to clarify what is selected and where the insertion position is.
>
> The status bar explicitly states the insertion point for a selected component.

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, do one of the following:

- Select the target position by clicking the node.
- 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 (in the Visual Editor) or horizontal line with an embedded up arrow (in the Structure window), 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 (in the 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. Table 20-5 summarizes the options.

Table 20-5 Target Position Options

Option	Description
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 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.

How to Insert XSD Components

In the XSD Visual Editor and Structure window 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.

> **Note:**
>
> Pasting multiple components that were copied from different places in the XML schema hierarchy can lead to unexpected results.

Inserting XSD Components Using the Components Window

You can insert XSD components by dragging from the Components window or by using a context menu. You can also select the target position in the visual editor or Structure window and then click the component in the Components window.

To insert XSD components using the Components Window:

1. In the XSD Visual Editor or Structure window, locate the 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 Components window, select an XSD component list from the dropdown list box, and then drag the component from the list and drop into the desired target position in the XSD Visual Editor or Structure window.

Inserting XSD Components Using the Context Menu

You can also right-click to display a context menu with options for inserting XSD components.

To insert XSD components using the context menu:

1. In the XSD Visual Editor or Structure window, right-click the 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.

How to Cut XSD Components

When you cut a component, it is removed from the editor and placed into a local clipboard accessible only by JDeveloper, not to the system clipboard. If you quit without pasting the component, the cut version of the component is lost.

You can cut, copy, and paste between files of the same project or different projects.

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 How to Delete XSD Components .

To cut one or more components:

1. Select the XSD component you want 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.

How to Copy XSD Components

You can copy XSD components in the visual editor or the Structure window. You can cut, copy, and paste between files of the same project or different projects.

To copy one or more components:

1. Select the XSD component in the visual editor.
2. Do one of the following:
   - Press Ctrl+C.
   - Right-click and select **Copy**.
   - Choose **Edit** > **Copy** from the main menu.
   - Hold down the Ctrl key and drag a copy of the selected component to a target position.

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. See How to Select XSD Components .

To delete one or more XSD components:

1. Select one or more XSD components to delete in the visual editor.
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.

How to Paste XSD Elements

The elements you cut or copy from the XSD Visual Editor or Structure window can be pasted into any other XSD file in the application. For more information, see How to Select Target Positions for XSD Components.

You can cut, copy, and paste between files of the same project or different projects.

To paste an element:

1. Open the file.
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**.

How to Move XSD Components

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.

You can move one or multiple components at a time. However, selecting and moving multiple, non-adjacent components or multiple components from different levels in the schema hierarchy can lead to unexpected results.

Moving Components by Dragging

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.

To move components by dragging:

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 How to Select Target Positions for XSD Components.
- Right-click and drag the component(s) from its original position to an insertion point. Choose **Move Nodes Here** from the context menu.

Moving Components by Cutting and Pasting

You can 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 How to Select XSD Components .

To move components by cutting and pasting:

Do either of the following:

- Cut the component(s). Then, paste it into some other position in the visual editor or Schema structure window.
- Cut the component(s). Then, paste it 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.

How to Set and Modify XSD Component Properties

The Properties window displays the properties of XSD components selected. Use the Properties window to set or modify the property values for any component in your XML Schema. Set property values are marked with a green square.

Choose **Edit** > **Undo Change Attribute** from the main menu.

To change a property back to its default setting, click on the down arrow next to the property, and choose **Reset to Default.**

To set a component's properties:

1. With an XML Schema open, select a component.

   The Properties window displays the property values. If the Properties window is not in view, choose **View** > **Properties window** or use the shortcut Ctrl+Shift+I.

2. Scroll until the property is visible, then select it with the mouse or arrow keys.

   A brief description of the property is displayed at the bottom of the Properties window.

> **Tip:**
>
> To quickly locate a property in a long list, click the search button in the Properties window toolbar. In the **Find** text field, enter the name of the property, then press Enter.

3. In the right column, enter one of the property values shown in Table 20-6.

Table 20-6 Property Values

Field	Description
Text field	Enter the string value for that property, for example a text value or a number value, then press `Enter`.
Value field with a down arrow	Click the down arrow and choose a value from the list, then press Enter
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.

How to Set Properties for Multiple XSD Components

If you have multiple components selected, by default the Properties window displays all the properties of the selected components. Click the **Union** button in the Properties window 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. Select and edit the desired property in the Properties window.

   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.

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.

For more information on XLIFF, see the OASIS open standard website at `https://www.oasis-open.org`.

To create a new XLIFF file, choose **File** menu >**New** > **From Gallery** > **General** > **XML** > **XML Localization File**.

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.

Developing Databound XML Pages with XSQL Servlet

Learn how to use XSQL Servlet, which lets you create and use XSQL pages as clients

You will find a complete development environment to simplify the task of developing databound XML pages with XSQL servlet. 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.

Table 20-7 shows the logical layers in an XSQL Servlet application.

Table 20-7 XSQL Servlet Logical Layers

Layer	Description
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 Servlet	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.

Supporting XSQL Servlet Clients

Support for XSQL Servlet includes the following:

- XSQL tags on the Components window
- Create XSQL pages automatically
- 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

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 you 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, the `emp.xsql` file below:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="emp.xsl"?>
<FAQ xmlns:xsql="urn:oracle-xsql" connection = "scott">
  <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 below:

```
<EMPLOYEES>
  <EMP>
    <ENAME>Scott</ENAME>
    <SAL>1000</SAL>
    <DEPARTMENT>Boston</DEPARTMENT>
  </EMP>
  <EMP>
...
  </EMP>
</EMPLOYEES>
```

For more information on XSQL Servlet, see your Oracle10*i* documentation.

How to Create an XSQL File

Using the wizard to create an XSQL file adds a skeleton 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 Components window, and modify the file with your own style sheet information.

To create an XSQL file:

1. In the Applications window, select the project in which you want to create the new XSQL page.
2. Choose **File** > **New** > **From Gallery**.
3. In the **Categories** tree, expand **General** and select **XML**.
4. In the Items list, double-click **XSQL File**.

How to Edit XML Files with XSQL Tags

The XML Editor supports syntax highlighting in the Structure window view, and the Properties window. You can also select tags from the Components window to insert in your pages while you are editing.

To use the XML Editor to edit an XSQL file:

1. In the Applications window, double-click right-click an XSQL file.
2. Select the Source tab of the editor.
3. Choose **View** > **Components window** to open the Components window and select the **XSQL** tag page from the dropdown list in the window. You can then select XSQL tags from the window.
4. 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.
5. 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.
6. While you are editing, or after you finish, you can right-click in the file and choose **Auto Indent XML** to correctly indent the file.
7. You can also right-click in any tag and choose **Locate in Structure** to highlight that tag in the Structure window.

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. Errors display in the XML Validation Errors tab of the Log.

To check the syntax in an XSQL file:

- In the Applications window, or in an open XML Editor window, right-click an XSQL file and choose **Validate XML**.

> **Note:**
>
> The **Validate XML** command on this context menu is disabled whenever an XML file does not have an XML namespace defined.

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. This procedure shows how to use the XSQL Query tag to display data.

To create an XSQL servlet client that directly accesses the database:

1. Select a project in the Applications window and choose **File** > **New** > **Gallery**.
2. In the **Categories** list, select **General** and select **XML**.

3. In the **Items** list, double-click **XSQL File**.

   This adds a skeleton XSQL file named `untitled#.xsql` to your project.

4. In the Applications window, double click the new XSQL file to open the editor.
5. Place your cursor in the blank line after the `<page xmlns:xsql="urn:oracle-xsql">` tag.
6. Choose **View** > **Components window** to open the Components window if it is not displayed.
7. Select **XSQL Tags** from the dropdown list in the Components window if it is not displayed.
8. Select the **Query** tag from the Components window.

   The **Query** tag executes a SQL statement and includes its result set in XML format

9. 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.
10. After entering attributes, click **Next**.
11. In the Connection Selection dialog, select your connection if it is not selected, then click **Next**.
12. 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.

13. Click **Finish**.

    Notice that the **Query** tag and attributes you entered appear in the XSQL page.

14. Choose **File** > **Save All t**o save your work thus far.
15. Right-click the XSQL file in the Applications window, 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.

Creating XSQL Servlet Clients for Business Components

You can create XML-based clients for business components using XSQL servlet. 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. Select a project in the Applications window and choose **File** > **New > From Gallery** 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 application.
5. Select the new project in the Applications window 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 Applications window, 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** > **Components window** to open the Components window if it is not displayed.
11. Select **XSQL tags** from the dropdown list in the Components window if it is not displayed.
12. Select the `ViewObject Show` tag from the Components window.

    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, then click **Next.**
14. Change or accept the default values for the 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 your work.
17. Right-click the XSQL file in the Applications window, 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:**
>
> 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.

What You May Need to Know About XSQL Error JBO-27122

You may get the XSQL error JBO-27122 while querying view objects with circular `ViewLink` accessors.

Consider a scenario in which there are foreign key relationships between multiple tables. For example, in the HR schema in the Oracle 10i Release 2 database samples, there is such a relationship between countries, departments, employees and locations tables. If you created a simple Business Components project on top of these tables and further created an XSQL client in which you access the view object called `CountriesView1`, you will get an error such as the one displayed in [the example below.

```
<?xml version="1.0" encoding="windows-1252" ?>
- <!--
| 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" ?>


-->
- <page>
```

```
- <xsql-error action="xsql:action">
  <message>JBO-27122: SQL error during statement preparation. Statement: SELECT
Employees.EMPLOYEE_ID, Employees.FIRST_NAME, Employees.LAST_NAME, Employees.EMAIL,
Employees.PHONE_NUMBER, Employees.HIRE_DATE, Employees.JOB_ID, Employees.SALARY,
Employees.COMMISSION_PCT, Employees.MANAGER_ID, Employees.DEPARTMENT_ID, Employees.DN
FROM EMPLOYEES Employees WHERE (Employees.DEPARTMENT_ID = :1)</message>
  </xsql-error>
  </page>
```

The reason you are getting this error is because of the way the underlying `writeXML()` method works in combination with the fact that you are using the default values for the max-levels parameter. This causes an infinite loop because of the circular references created by the foreign keys as mentioned above - internally causing an `ORA-1000 MaxOpenCursors` exceeded error.

The way to work around this is to modify your code as shown below:

```
<xsql:action handler="oracle.jbo.xsql.ViewObject"
      name="YourViewUsageName"
      appmodule="a.b.c.YourModuleName"
      configname="YourModuleNameLocal">
   <view-attribute-list viewdefname="a.b.c.YourViewName"
         include-only"Attr1 Attr2 Attr3"
       :
</xsql:action>
```

You can have multiple `<view-attribute-list>` elements for each view definition that you want to control the attribute list for. As soon as you have one `<view-attribute-list>` element, then use the new API for `writeXML()` that will only include the attributes that are listed in the `<view-attribute-list>` elements.

In order to show the details for a VO, you need to include the name of the view link accessory attribute in the list of attribute names in the "include-only" list for that view definition. The following example contains a working example based on the `CountriesView`.

```
<?xml version="1.0" encoding='windows-1252'?>
<!--
| 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" ?>
-->
<page xmlns:xsql="urn:oracle-xsql">
    <xsql:action
    handler="oracle.jbo.xsql.ViewObject"
    name="CountriesView1"
    configname="HrModuleLocal"
    appmodule="hr.HrModule">
    <view-attribute-list viewdefname="hr.HrModule.CountriesView"
    include-only="CountryId CountryName CurrencyName" />
    </xsql:action>

</page>
```

How to Create 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, and so on. 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.

```
// 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());
    }
}
```

How to Deploy XSQL Servlets

XSQL servlet generates executable packages that contain the information and logic required to deploy the database to the servlet client environments. The database deployment package can be incorporated into a setup and deployment solution or can be shipped to clients as a separate application. When you deploy an XSQL servlet, you must specify a master database, the deployment scenario, and then generate a self-contained executable that is ready for deployment.

The following is a custom application. As you create it, you will add appropriate features for developing an XSQL application (that it, when you choose Java and XSQL as part of creating the application).

To deploy an XSQL servlet:

1. If necessary, create a new application and project.

   For more information, see Creating Applications and Projects.
2. If the XSQL file is not already open in the source editor, in the Applications window, double-click the name of the XSQL file you just created to open it.
3. Open the New Gallery by choosing **File** > **New**.
4. In the New Gallery, in the Categories tree, under General, select **XML**.

5. In the Items list, double-click **XSQL File**.
6. In the Components window, drag and drop Query (XSQL) onto the page. This opens a Query wizard, with information about the db connection and enter the SQL query. When you finish the wizard, it adds a `<xsql:query>` tag to the XSQL file.
7. Click **Next**. On the second page of the wizard, if you have not already created a database connection, create a new one. Otherwise choose the existing database connection to use.

   The schema and tables you plan to query should be in the database. For more information about setting up a database connection, see Configuring Database Connections. Click **Next**.
8. Add a SQL query to the file, for example, the query shown below:

   ```
   SELECT DISTINCT d.department_id as h_deptno, department_name as "Department_name"
   FROM departments d, employees e
   WHERE d.department_id = e.department_id
   ORDER BY d. department_name
   ```
9. Click **Finish**. The query and associated XSQL tags are entered in the XSQL file.
10. Now 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 Running Java EE Applications in the Integrated Application Server.
11. In the context menu of the XSQL file in the Applications window or in the source editor, click **Run**. If necessary, the Integrated WebLogic Server will create the default domain and start. 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.

    When you click **OK** in the dialog the default domain is created. This may take a few minutes and you can follow the progress in the Log window.

    When the XSQL file runs in Integrated WebLogic Server, the default browser displays the results of the SQL query.
12. Once you have tested the XSQL file successfully in Integrated WebLogic Server, the next step is to deploy and run the application in an application server.
13. 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:

    `connection="java:comp/env/jdbc/`*`database-connection-name`*`DS"`

    with

    `connection="jdbc/`*`database-connection-name`*`DS"`

    > **Note:**
    >
    > If you want to run the application in the Integrated WebLogic Server, you need to change the connection information back again.

14. In order to deploy the application, you first have to create a deployment profile and deploy the application to it. In the Applications window, right-click the project containing your XSQL servlet, then choose **New**. In the New Gallery, expand **General** and select **Deployment Profiles**.

15. Choose a profile, for example, a WAR deployment profile and click **OK** and continue to create the deployment profile. For more information, see How to Create and Edit Deployment Profiles.
16. 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**.
17. 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 Setting Up JDBC Data Sources on .
    - Create a connection to the application server. For more information, see 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`.

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 Applications window, right-click and choose **Run** 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.

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 Applications window, select the project in which you want to create the new XSL file.
2. Choose **File** > **New** > **From Gallery**.
3. In the **Categories** tree, expand **General** and select **XML**.
4. In the **Items** list, double-click **XSL Style Sheet**.
5. In the **File Name** field, enter the name of the file you want to generate.

6. Leave the **Directory Name** field unchanged to save your work in the directory where JDeveloper expects to find web application files.

   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.

   The style sheet in the example below demonstrates the conversion of XML to WML. It uses the default `DeptView` in a BC4J application.

```
<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>
```

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 Applications window, double-click the XSQL file to which you want to add a style sheet.
2. In the source file, 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 Applications window, 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 Applications window and opens in the XML Editor. You can close the open files.
8. Right-click the XSQL file in the Applications window and choose **Run** to open the file in your web browser.

   You can see the formatted XML data in the browser.

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 Applications window, 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.

Using XML Metadata Properties in XSQL Files

The custom properties shown in Table 20-8 affect XML generation when using the `writeXML` method of a view object or row.

Table 20-8 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

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

produces 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>
```

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`

produces 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>
```

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.

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.

21

Developing and Securing Web Services

This chapter describes how JDeveloper enables you to develop, deploy, test, and monitor web services; secure web services using policies; and manage the Web Service Definition Language (WSDL) files. Learn how to discover web services using Universal Description, Discovery and Integration (UDDI) and create web service clients.
This chapter includes the following sections:

- About Developing and Securing Web Services
- Using JDeveloper to Create and Use Web Services
- Working with Web Services in a UDDI Registry
- Creating JAX-WS Web Services and Clients
- Creating RESTful Web Services and Clients
- Creating WebSockets
- Attaching Policies
- Deploying Web Services
- Testing and Debugging Web Services
- Monitoring Web Services Network Traffic

About Developing and Securing Web Services

Use JDeveloper to develop 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.

If you use web services in your application, you use JDeveloper to perform the following tasks:

- 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 Universal Description, Discovery and Integration (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 web services from the underlying Java implementation (bottom up)
  - Create Simple Object Access Protocol (SOAP) web services from the WSDL (top-down)
- Manage WSDL files for SOAP services
- Secure web services using policies
- Test and debug web services

- Deploy web services to the Integrated WebLogic Server or Oracle WebLogic Server
- Monitor and analyze deploy web services
- Publish web services to a UDDI registry

The following sections provide more information about developing and securing web services using JDeveloper:

- Developing Java EE Web Services Using JDeveloper
- Securing Java EE Web Services Using JDeveloper
- Discovering and Using Web Services

Developing Java EE Web Services Using JDeveloper

JDeveloper supports the web service technologies and standards defined in Table 21-1 for developing and securing Java EE web services.

Table 21-1 Java EE Web Service Technologies and Standards Supported by JDeveloper

Web Service Technology	Web Service Standard	Description
Simple Object Access Protocol (SOAP)	Java API for XML-Based Web services (JAX-WS) 2.2	You can create SOAP web services, using JAX-WS, from Java classes and the remote interface of EJBs. 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. Alternatively, you can create a JAX-WS web service starting with a WSDL, as a top-down web service. The JAX-WS implementation in WebLogic Server is extended from the JAX-WS Reference Implementation (RI) developed by the Glassfish Community. See Creating JAX-WS Web Services and Clients.

Table 21-1 (Cont.) Java EE Web Service Technologies and Standards Supported by JDeveloper

Web Service Technology	Web Service Standard	Description
Representational State Transfer (REST)	Java API for RESTful Web Services (JAX-RS) 2.0	REST describes any simple interface that transmits data over a standardized interface (such as HTTP) without an additional messaging layer, such as SOAP. Using JAX-RS, you can develop web services that are based on REST, referred to as “RESTful web services," from Java classes. WebLogic Server supports the following JAX-RS Reference Implementations (RIs): • **Jersey 2.5.1 (JAX-RS 2.0 RI)**—This implementation is offered as a shared library and provides a production quality implementation of the JSR-339 JAX-RS 2.0 specification, defined at: `http://jcp.org/en/jsr/detail?id=339`. The Jersey 2.5.1 (JAX-RS 2.0 RI) shared library is auto-deployed to the Integrated WebLogic Server. **Note**: RESTful web services and clients that are built using Jersey 2.5.1 (JAX-RS 2.0 RI) are secured using OWSM policies. For more information about securing RESTful web services and clients built using Jersey 2.5.1 (JAX-RS 2.0 RI), see "Securing RESTful Web Services and Clients". See Creating RESTful Web Services and Clients. **Note:** Support for the Jersey 1.18 (JAX-RS 1.1RI) client APIs of WebLogic Server is deprecated. However, support is maintained for backward compatibility.

For guidelines to consider when choosing between SOAP and REST technologies, see How Do I Choose Between SOAP and REST? in *Understanding WebLogic Web Services for Oracle WebLogic Server*.

Securing Java EE Web Services Using JDeveloper

To secure Java EE web services using JDeveloper you can attach one of the policy types defined in the following table.

Table 21-2 Types of Policies for Securing Java EE Web Services

Type of Policy	Description
Oracle Web Services Manager (OWSM) Policy	Policy provided by OWSM. See *Understanding Oracle Web Services Manager*. You can attach OWSM *security* policies only to Java EE web services. You manage OWSM policies from Oracle Enterprise Manager Fusion Middleware Control. See *Securing Web Services and Managing Policies with Oracle Web Services Manager*.
WebLogic Web Service Policy	Policy provided by WebLogic Server. You can attach WebLogic web service policies to JAX-WS web services only. See *Securing WebLogic Web Services for Oracle WebLogic Server*. You manage WebLogic web service policies from WebLogic Administration Console.

Note:

For Java EE 7, Java API for RESTful Web Services (JAX-RS) 2.0 adds support of the Jersey 2.5.1 Java API for RESTful Web Services (JAX-RS) 2.0 Reference Implementation (RI) as a pre-built shared library. RESTful web services and clients that were built using Jersey 2.5.1 (JAX-RS 2.0 RI) are secured with Oracle Web Services Manager (OWSM) security policies.

It is recommended that you use OWSM policies over WebLogic web service policies whenever possible. You cannot mix your use of OWSM and WebLogic web service policies on the same web service.

Discovering and Using Web Services

You can quickly create a client to an existing web service in order to use it in your application. For more information, see:

- How to Create JAX-WS Web Service Clients
- How to Create RESTful Web Service Clients

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 Working with Web Services in a UDDI Registry.

Using JDeveloper to Create and Use Web Services

Learn about JDeveloper's features and tools which help you to create 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.

- How to Use Proxy Settings and JDeveloper
- How to Set the Context Root for Web Services
- How to Configure Connections to Use with Web Services
- How to Work with Type Mappings

- How to Choose Your Deployment Platform
- How to Work with Web Services Code Insight

How to Use Proxy Settings and JDeveloper

By default, JDeveloper does not use a proxy when connecting to the Internet. If you have problems making connections from JDeveloper, you may need to change the proxy server settings you use.

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.

> **Note:**
>
> JDeveloper does not support NTLM proxy servers.

The following sections describe how to enable and disable proxy setting using JDeveloper:

- Using the Default Browser Proxy Settings
- Configuring Custom Proxy Settings
- Disabling the Use of a Proxy Server When Accessing the Internet

Using the Default Browser Proxy Settings

You may find it convenient to use the proxy settings that are configured for the default browser.

To use the default browser proxy settings:

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. Select the **Web Browsers** tab and select the desired default browser from the list.
3. Select the **Proxy Settings** tab to configure proxy settings.
4. Select **Use System Default Proxy Settings**.
5. Click **OK**.

Configuring Custom Proxy Settings

You can configure custom proxy settings. This is useful if you need to exclude specific IP addresses or host names from the list.

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.

To configure custom proxy setting:

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. Select the **Proxy Settings** tab to configure proxy settings.
3. Select **Manual Proxy Settings** and enter the host and port name of the proxy server.
4. List any host names or IP addresses that you want to exclude in the **No Proxy for** field.
5. Click **OK**.

Disabling the Use of a Proxy Server When Accessing the Internet

You can disable the use of a proxy when accessing the Internet. This is the default behavior.

To disable the use of a proxy server when accessing the Internet:

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. Select the **Proxy Settings** tab to configure proxy settings.
3. Select **No Proxy**.
4. Click **OK**.

How to Set the Context Root for Web Services

The context root (also referred to as context path) 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.

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`.

At the project level, you can set the context root that will be assigned to the deployed Java EE web application on Integrated WebLogic Server. The context root value defaults to:

applicationname-projectname`-context-root`

To set the context root for web services:

1. In the Applications window, 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. Select **Java EE Application**.
3. Select whether you want to use personal project settings or common project settings:
   - To use personal project settings, select **Use Custom Settings** and click the **Customize Settings** button.
   - To use common project settings, select **Use Project Settings**.
4. Update the **Java EE Web Context Root** field to define the context root that will be assigned to the application when running or deploying the contents of the project as a Java EE web application on Integrated WebLogic Server.
5. Click **OK**.

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 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 How to Define UDDI Registry Connections.

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 objects 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://www.oracle.com/technetwork/java/javase/tech/index-jsp-138795.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. For a list of XML Schema data types and their corresponding Java data types, see “XML-to-Java Mapping for Built-in Data Types" in *Developing JAX-WS Web Services for Oracle WebLogic Server*. In addition, a service method can accept and return a single piece of XML element data, passed as an `org.w3c.dom.Element`.

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.

You can use the JAXB binding language to define custom binding declarations or specify JAXB annotations to control the conversion of data between XML and Java.

WebLogic Server provides two data binding and JAXB providers:

- EclipseLink MOXy, the default in this release of WebLogic Server, is a fully compliant JAXB implementation. In addition to offering the standard JAXB features, EclipseLink MOXy provides useful extensions, such as the ability to use an external metadata file to configure the equivalent of JAXB annotations without modifying the Java source it refers to, and

XPath based mapping. The JAXB enhancements can be used in the annotations on a service endpoint interface (SEI) or one of the value types used by the SEI. Users of JAXB in standalone mode can also take advantage of these features.

Some of the additional extensions offered by EclipseLink MOXy include:

- Extensions for mapping JPA entities to XML
- Bidirectional mapping
- Virtual properties
- Ability to bootstrap from metadata and generate in-memory domain classes (Dynamic MOXy)

For a web service, the EclipseLink MOXy extensions can be leveraged on the server side only, and only in the Java to WSDL scenario, in which the SEI and value types can use the extended EclipseLink functionality. For more information about these extensions and EclipseLink MOXy, see *The EclipseLink MOXy (JAXB) User's Guide* at `http://wiki.eclipse.org/EclipseLink/UserGuide/MOXy`.

No configuration is required to use the EclipseLink MOXy providers.

- Glassfish RI JAXB, which is the default Glassfish JAXB implementation, and was the default JAXB offering in WebLogic Server in previous releases. The Glassfish RI JAXB proprietary features will not work with EclipseLink MOXy. If desired, you can enable the Glassfish RI JAXB data binding and JAXB providers at the server or application level. For more information, see Using the Glassfish RI JAXB Data Binding and JAXB Providers.

For more information about using JAXB with JAX-WS web services, see Using JAXB Data Binding.

How to Choose Your Deployment Platform

The first time you create a web service in a project this dialog appears, and the version you select is used for all web services you create in the project.

For SOAP web services, this dialog also appears when you right-click an existing WSDL and select **Create Web Service**, and then choose to create a new web service project from that WSDL.

The first time you create a web service in a project, you are offered a choice of deployment platforms, as defined in Table 21-3 andTable 21-4 for SOAP and RESTful web services, respectively. 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.

> **Note:**
>
> Java EE 7 is supported.

Table 21-3 lists the deployment platforms available for SOAP web services.

Table 21-3 Deployment Platforms for SOAP Web Services

Deployment Platform	Description
Java EE 6 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. 6. This option provides support for deploying to WebLogic Server with Java annotations using the JAX-WS annotation specification.
Java EE 6 with support for JAX-WS RI	Generates a JAX-WS web service for deploying to any container that supports the JAX-WS Reference Implementation.

Table 21-4 lists the deployment platforms available for RESTful web services.

Table 21-4 Deployment Platforms for RESTful Web Services

Deployment Platform	Description
JAX-RS 1.0 Style	Generates a RESTful web service that is compatible with Jersey 1.18 (JAX-RS 1.1 RI).
JAX-RS 2.0 Style	Generates a RESTful web service that is compatible with Jersey 2.5.1 (JAX-RS 2.0 RI). The Jersey 2.5.1 (JAX-RS 2.0 RI) shared library is auto-deployed to the Integrated WebLogic Server. **Note**: You can attach OWSM policies to RESTful web services and clients that are built using Jersey 1.18 (JAX-RS 1.1 RI) RESTful web services and clients that are built using Jersey 2.5.1 (JAX-RS 2.0 RI) are secured using OWSM policies. For more information about securing RESTful web services and clients built using Jersey 2.5.1 (JAX-RS 2.0 RI), see "Securing RESTful Web Services and Clients" in *Developing and Securing RESTful Web Services for Oracle WebLogic Server*.

How to Work with Web Services Code Insight

When typing annotations in a Java class, the web services Code Insight completes annotations. The tool is also available for WSDL documents in the XML editor (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 > Preferences > Code Editor > Code Insight**.

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 6, 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 6, with support for JAX-WS RI. In this case, JDeveloper adds:
  - `import javax.jws.WebService;` statement to the class
  - `sun-jaxws.xml` and `web.xml` files to the project

> **Note:**
>
> Java EE 7 is supported.

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 the Java class, click the **Audit fix** icon and select how you would like to reconcile the discrepancy.

Working with Web Services in a UDDI Registry

Learn about UDDI registries and how you can work with them.

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 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:

- How to Define UDDI Registry Connections
- What You May Need to Know About Choosing the View for your UDDI Registry Connection
- How to Search for Web Services in a UDDI Registry
- How to Generate Proxies to Use Web Services Located in a UDDI Registry
- How to Display Reports of Web Services Located in a UDDI Registry
- How to Publish Web Services to a UDDI Registry

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.

- Creating UDDI Registry Connections
- Editing the UDDI Registry Connections
- Changing the View of UDDI Registry Connections
- Refreshing the UDDI Registry Connections
- Deleting the UDDI Registry Connections

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 Resources window, in the Connections panel.

To create a UDDI registry connection:

1. In the Applications window, select the project.
2. Choose **File > New > From Gallery** to open the New Gallery.
3. In the **Categories** tree, expand **Business Tier** and select **Web Services**.
4. In the **Items** list, select **UDDI Registry Connection** and click **OK** 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.

Editing the UDDI Registry Connections

You can edit an existing UDDI registry connection. For example, to change the name of the connection or the URL of the inquiry endpoint.

To edit the UDDI registry connection:

1. In the main menu, choose **Window > Resources**. By default, the Resources window is displayed to the right of the JDeveloper window.
2. In the Resources window, under IDE Connections, expand **UDDI Registry**.
3. From the context menu of the UDDI registry connection you want to edit, choose **Properties**.

   The Edit UDDI Registry Connection wizard is launched.

   For more information at any time, press **F1** or click **Help** from within the Edit UDDI Registry Connection wizard.

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 What You May Need to Know About Choosing the View for your UDDI Registry Connection.

To change the view of UDDI registry connections:

1. In the main menu, choose **Window > Resources**. By default, the Resources window is displayed to the right of the JDeveloper window.
2. In the Resources window, under IDE Connections, expand **UDDI Registry**.
3. From the context menu of the UDDI registry connection you want to edit, choose **Render Business Perspective** or **Render Category Perspective**.

Refreshing the UDDI Registry Connections

You can refresh a UDDI registry connection to ensure that information stored under the connection is up-to-date.

To refresh the UDDI registry connection:

1. In the main menu, choose **Window > Resources**. By default, the Resources window is displayed to the right of the JDeveloper window.
2. In the Resources window, under IDE Connections, expand **UDDI Registry**.
3. From the context menu of the UDDI registry connection you want to refresh, choose **Refresh**.

Deleting the UDDI Registry Connections

When no longer needed, you can delete a UDDI registry connection from the Resources window.

To delete a UDDI registry connection:

1. In the main menu, choose **Window > Resources**. By default, the Resources window is displayed to the right of the JDeveloper window.
2. In the Resources window, under IDE Connections, expand **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**.

What You May Need to Know About Choosing the View for your UDDI Registry Connection

When you create the connection, as described in 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.

Choosing the Business View

A UDDI registry contains four data structure types that group information about web services:

- **Business Entities**: 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 Resources window.
- **Business Services**: Contains descriptive information for a family of services, including the name and brief description, and category information.
- **Binding Templates**: 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.

Choosing Category View

If you choose **Category View**, you can search for web services based on one or more of the following categories:

- **ISO 3166**: Search by location using the International Organization for Standardization (ISO) 3166 standard codes.
- **NAICS**: Specify the type of industry using the North American Industry Classification System (NAICS).
- **SIC**: Specify the type of industry using the Standard Industrial Classification (SIC).
- **UDDI Types**: Search by UDDI type.
- **UDDI WSDL Types**: Search by UDDI WSDL type.
- **UNSPSC**: Search by type of service using the United Nations Standard Products and Services Code (UNSPC).

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.

How to Search for Web Services in a UDDI Registry

You can search a UDDI registry connection in the Resources window 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 Creating UDDI Registry Connections.
2. In the Resources window, search for the web service. For more information, see Working with the Resources Window.

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 Resources window.

> **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 to use web services located in a UDDI registry:

1. Open the Resources window.

   In the main menu, choose **Window > Resources**. By default, the Resources window 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 endpoint (port).
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.

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 web service located in a UDDI registry:

1. Open the Resources window.

   In the main menu, choose **Window > Resources**. By default, the Resources window is displayed to the right of the JDeveloper window.
2. Navigate to the web service you want, or search for it.
3. Right-click the service, and choose **View Report**.

   A report of the web service is displayed in the source editor.

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 window. 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 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 window, 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.

Creating JAX-WS Web Services and Clients

Java API for XML Web Services (JAX-WS) is a technology for building web services and clients that communicate using XML. Learn how to create webservices and clients using JDeveloper.

To create JAX-WS web services and clients using JDeveloper, you can:

- Create JAX-WS web services from Java classes and the remote interface of EJBs, as described in How to Create JAX-WS Web Services (Bottom-up).
- Create a JAX-WS web service starting with a WSDL, as a top-down web service, as described in How to Create JAX-WS Web Services from WSDL (Top-down).
- Create a JAX-WS client and proxy classes to access a service using the Create Web Service Client and Proxy wizard, as described in How to Create JAX-WS Web Service Clients.
- Enable atomic transactions for a web service implementation, as described in How to Use Web Service Atomic Transactions.
- Use SOAP over JMS transport to communicate using JMS destinations instead of HTTP connections, as described in How to Use SOAP Over JMS Transport.
- Enable Fast Infoset for a web service implementation, as described in How to Use Fast Infoset for Optimizing XML Transmission.
- Enable MTOM for a web service implementation, as described in How to Use MTOM for Optimizing Binary Transmission.
- Create, display, and save WSDL files, as described in How to Manage WSDL Files.
- Edit or delete JAX-WS web services, as described in How to Edit JAX-WS Web Services and How to Delete JAX-WS Web Services, respectively.

How to Create JAX-WS 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.

The following sections describe how to generate different types of web services from the bottom up:

- Creating Java Web Services
- Using Web Service Annotations
- Using the Properties Window
- Creating Database Web Service Providers
- Regenerating Web Services from Source
- Using Handlers

- Handling Overloaded Methods

For information about:

- Using top-down development—starting from the WSDL—see How to Create JAX-WS Web Services from WSDL (Top-down).
- Using web services atomic transactions, see How to Use Web Service Atomic Transactions.
- Using SOAP over JMS transport, see How to Use SOAP Over JMS Transport.

Creating Java Web Services

You can create web services from:

- Java classes
- Remote interface of EJBs

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 How to Set the Context Root for Web Services.

To create the Java web service:

1. In the Applications window, select the project containing the Java class or EJB from which you want to create a web service.
2. Choose **File > New > From Gallery** to open the New Gallery.
3. In the **Categories** list, expand **Business Tier** and select **Web Services**.
4. In the **Items** list, select **Java Web Service** and click **OK** to launch the Create Java Web Service wizard.

   For detailed help about completing the wizard, press **F1** or click **Help** from within the wizard.

Alternatively, you can launch the Create Java Web Service wizard by right-clicking on the Java class from which you want to create a web service and selecting **Create Web service**.

> **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. For more information, see How to Choose Your Deployment Platform.
>
> When using the Create Java Web Service wizard, if you enter a class name for a Java class that does not exist, the wizard provides the option to generate a default Java class, with a single `String helloWorld(String)` method, to be used as the implementation class for the web service. If you decline to generate the default Java class, you will be prompted to select a valid class name.

Using Web Service Annotations

The JSR-181 specification specifies web services metadata, which allows you to use annotations 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 endpoints.

You can add 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.

For more information, refer to:

- JSR-181 specification (`http://jcp.org/en/jsr/detail?id=18`)
- JAX-WS specification (`http://jcp.org/aboutJava/communityprocess/mrel/jsr224/index2.html`)
- For JWS annotations available with WebLogic Server see “JWS Annotation Reference".

> **Note:**
>
> If you delete the annotations using the Edit Web Services dialog, any annotations that you entered manually are also deleted.

To use web service annotations:

1. Open the Java class in the source editor.
2. On the line where you want to add the annotation, type `@` and pause for a couple of seconds.

   Code Insight displays possible values.

Using the Properties Window

You can add the @WebService annotation and supporting files to your web service project automatically using the Properties window.

To create a JAX-WS web service in the Properties window:

1. With the web service class open in the source editor, choose **Window > Properties** to open the Properties window.

   For more information at any time, press **F1** or click **Help** from within the Properties window.
2. With the cursor in the public class, navigate to the JAX-WS node in the Properties window.
3. Select **Web Service Bean Class**.

   The Select Deployment Plan Platform dialog box is displayed. For information about selecting the deployment platform, see How to Choose Your Deployment Platform.
4. Select a deployment platform and click **OK**.

   The `javax.jws.WebService` annotation is imported and added to the public class of the web service, and the required deployment files (for example, `web.xml`) are added to your project.

Creating 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 Deploying Web Services.

It should be noted that:

- The wizard generates a JAX-WS web service Provider.
- If you edit the 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.

To create the TopLink DB web service Provider from a project:

1. In the Applications window, select the project.
2. Choose **File > New > From Gallery** 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.

Regenerating 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 when not created using a TopLink DB Web Service Provider:

1. In the Applications window, 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.

> **Note:**
>
> If the Web Service was originally created using TopLink DB Web Service Provider, you must recreate the service provider. To do so, rerun the wizard and select the desired service provider option from the New menu. JDeveloper will prompt for a confirmation to overwrite the previously generated artifacts. Continue with the TopLink web service provider wizard to generate a new TopLink web service provider which will replace the one that currently exists.

Using Handlers

JDeveloper allows you to specify the handler classes to edit with the web service message. The handlers can use initialized parameters, SOAP roles, or SOAP headers. See Creating and Using SOAP Message Handlers.

To define handlers:

1. Create a web service or open the web service editor.
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.

Handling Overloaded Methods

If the Java class on which you base a web service has overloaded methods, JDeveloper handles them automatically.

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 *Developing JAX-WS Web Services for Oracle WebLogic Server*.

How to Create JAX-WS 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 locate a web service in a UDDI registry connection in the Resources window.

To create a JAX-WS web service from WSDL (top-down):

1. In the Applications window, select the project in which you want to create the web service.
2. Choose **File > New > From Gallery** 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 JAX-WS web service is created and the Java implementation class is opened automatically in the editor.

> **Note:**
>
> You can also create a web service from a WSDL by right-clicking an existing WSDL and selecting **Create Web Service** from the context menu. You will also have the option to create a new web service project from that WSDL

How to Create JAX-WS Web Service Clients

JDeveloper makes it easy to use a web service in your application by allowing you to create JAX-WS 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 “Developing Basic JAX-WS Web Service Clients".
- Administering web services and clients, see *Administering Web Services*.

The following sections describe how to create and use JAX-WS web service clients:

- Creating the Client and Proxy Classes

- Developing a JAX-WS Web Service Client Application
- Referencing Web Services Using the @WebServiceRef Annotation
- Regenerating Web Service Client and Proxy Classes
- Editing the Web Service Clients
- Deleting the Web Service Clients

See also the following sections describing how to view and manage the WSDL used to create the web service client:

- Viewing the WSDL Used to Create the Web Service Client
- Refreshing the Local Copy of the WSDL and Regenerating the Web Service Client Proxy and Classes
- Updating the Web Service WSDL Used by the Client at Run Time

Creating 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, attach policies, and define SOAP handlers, 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) and service interface for each exposed port and lists them in the Applications window. 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.

Creating Client and Proxy Classes to Access a Web Service

You can generate client and proxy classes for a web service that is defined outside of JDeveloper by launching the Create Web Service Client Proxy wizard and specifying the WSDL for the web service.

To create client and proxy classes to access a web service:

1. In the Applications window, select the project you want to use.
2. Choose **File > New > From Gallery** to open the New Gallery.

3. In the **Categories** list, select **Web Services**.
4. In the **Items** list, double-click **Web Service Client and Proxy** to launch 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.

Creating Client and Proxy Classes to Access a Web Service Defined in JDeveloper

You can generate client and proxy classes for a web service that is currently defined in JDeveloper from the Applications window.

To create a client and proxy classes to access a web service defined in JDeveloper:

Right-click the web service container in the Applications window, and choose **Create Client for Web Service Annotations**.

The Create Web Service Client and Proxy wizard opens and is pre-populated 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.

Developing a JAX-WS Web Service Client Application

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:

- Standalone 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 Referencing Web Services Using the @WebServiceRef Annotation

Developing a Standalone Client Application

A standalone 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 develop a standalone client application:

1. Open the client proxy class, called *port_name*`Client.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 Applications window.

2. Locate the comment `// Add your code to call the desired methods`, and add the appropriate code to invoke the web service.
3. Run the client.

Developing a 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 develop a Java SE client application:

1. Copy the generated client proxy classes to your Java SE application source directory.
2. Using the main client proxy class, called *port_name*`Client.java`, as your guide, add appropriate methods to access the web service from your application.
3. Run the application.

Developing a Java EE Component Client Application Deployed to WebLogic Server

This type of web service runs inside a Java Platform, Enterprise Edition (Java EE) Version 6 component deployed to WebLogic Server, such as an EJB, servlet, or another web service. A JEE web service client application, therefore, runs inside a WebLogic Server container.

To develop a Java EE component client application deployed to WebLogic Server:

1. Open the main client proxy class, called *port_name*`Client.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 Applications window.
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.

Referencing 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 the @WebServiceRef Annotation in *Developing JAX-WS Web Services for Oracle WebLogic Server*.

Enabling Web Service Atomic Transactions in a Web Service Client

For more information about web service atomic transactions, see How to Use Web Service Atomic Transactions.

You can enable web service atomic transactions in a web service client's injectable target.

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 21-5.

Regenerating Web Service Client and Proxy Classes

There are times that you may need to regenerate the web service client and proxy classes. For example, if the web service has been updated since they were last generated.

> **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:

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 Applications window, 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.

Editing the Web Service Clients

You can edit a web service client using the Web Service Client and Proxy editor.

To edit a web service client:

- Double-click on the web service client container within the Applications window.
- Right-click on the client within the Applications window, and select **Properties...**.

For help in completing the wizard, press **F1** or click **Help** from within the wizard.

Deleting the Web Service Clients

Once no longer needed, you can delete web service clients.

To delete a web service client:

1. In the Applications window, right-click on the web service client container, and select **Delete**.

   The Delete Proxy? dialog displays.

2. Ensure that the appropriate files are selected in the dialog box. Click **Select All** or **Deselect All** to select or deselect all proxy files.
3. Choose **OK**.

   The files are permanently erased.

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 Services Atomic Transactions.

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 Enabling Web Service Atomic Transactions in a Java Class.
- Using the Properties window, as described in Enabling Web Service Atomic Transactions in the Properties Window .

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 Services Atomic Transactions.

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 `@TransactionAttribute` annotation.

Table 21-5 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`
`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`

You can enable web service atomic transactions from a Java class, the Properties window, or a web service client's injectable target, as described in the following sections.

- Enabling Web Service Atomic Transactions in a Java Class
- Enabling Web Service Atomic Transactions in the Properties Window
- Enabling Web Service Atomic Transactions in a Web Service Client's Injectable Target

Enabling Web Service Atomic Transactions in a Java Class

You can enable web service atomic transactions in a Java class.

To enable web service atomic transactions in the Java class:

1. Open the web service class in the source editor.
2. 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])
```

Enabling Web Service Atomic Transactions in the Properties Window

You can enable web service atomic transactions in the Properties window.

To enable web service atomic transactions in the Properties window:

1. With the web service class open in the source editor, choose **Window > Properties** to open the Properties window.

   For more information at any time, press **F1** or click **Help** from within the Properties window.
2. With the cursor in the public class, `@WebService`, or two-way method line of the class, navigate to the JAX-WS Extensions node in the Properties window.
3. Select **Add Transactional**.

   The Properties window is refreshed to display options to enable or disable the feature, and to set the flow type and version. For more information about the configuration options, see Table 21-5.
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`.

The `@Transactional` annotation is imported and added to the public class.

Enabling Web Service Atomic Transactions in a Web Service Client's Injectable Target

You can enable web service atomic transactions in a web service client's injectable target.

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 21-5.

How to Use SOAP Over JMS Transport

Typically, web services and clients communicate using SOAP over HTTP/S as the connection protocol. You can, however, configure a WebLogic web service so that client applications use JMS as the transport.

Using SOAP over JMS transport, web services and clients communicate using JMS destinations instead of HTTP connections, offering the following benefits:

- Reliability
- Scalability
- Quality of service

As with web service reliable messaging, if WebLogic Server goes down while the method invocation is still in the queue, it will be handled as soon as WebLogic Server is restarted. When a client invokes a web service, the client does not wait for a response, and the execution of the client can continue. Using SOAP over JMS transport does require slightly more overhead and programming complexity than HTTP/S.

For each transport that you specify, WebLogic Server generates an additional port in the WSDL. For this reason, if you want to give client applications a choice of transports they can use when they invoke the web service (JMS, HTTP, or HTTPS), you should explicitly configure each transport.

> **Note:**
>
> SOAP over JMS transport is not compatible with the following web service features: reliable messaging and HTTP transport-specific security.

You can enable SOAP over JMS transport from a Java class, the Properties window, or a web service client, as described in the following sections.

- Developing Web Services Using JMS Transport
- Enabling JMS Transport in the Properties Window
- Developing Web Service Clients Using JMS Transport

For more information about SOAP over JMS transport, see Using SOAP Over JMS Transport in *Developing JAX-WS Web Services for Oracle WebLogic Server*.

Developing Web Services Using JMS Transport

To develop web services using JMS transport, use one of the following methods:

- Adding `@com.oracle.webservices.api.jms.JMSTransportService` annotation directly in the Java class, as described in “Using the @JMSTransportService Annotation" in *Developing JAX-WS Web Services for Oracle WebLogic Server*.
- Using the Properties window, as described in Enabling JMS Transport in the Properties Window.

Enabling JMS Transport in the Properties Window

To simplify configuration, you can enable JMS transport in the Properties window.

To enable JMS transport in the Properties window:

1. With the web service class open in the source editor, choose **Window > Properties** to open the Properties window.

   For more information at any time, press **F1** or click **Help** from within the Properties window.
2. With the cursor in the public class, `@WebService`, or two-way method line of the class, navigate to the JMS node in the Properties window.
3. Select **JMS**.

   The Properties window is refreshed to display options to configure SOAP over JMS transport. For more information about the configuration options, see Table 21-6.
4. Configure the JMS transport properties, as required.

When enabling JMS transport, you can configure the properties defined in the following table.

Table 21-6 Configuration Properties for SOAP Over JMS Transport

Name	Description
Activation Properties	Activation configuration properties passed to the JMS provider. To edit the activation properties, click **...** to open the Edit Property: Activation Properties dialog and specify values in the Value column for the activation properties. For a list of activation properties that are supported, see Configuring JMS Transport Properties.
Binding Version	Version of the SOAP JMS binding. This value must be set to `SOAP JMS 1.0` for this release, which equates to `com.oracle.webservices.api.jms.JMSConstants.SOAP11_JMS_BINDING`. This value maps to the `SOAPJMS_bindingVersion` JMS message property.
Delivery Mode	Delivery mode indicating whether the request message is persistent. Valid values are `Persistent` and `Non-Persistent`. This value defaults to `Persistent`.
Destination Name	JNDI name of the destination queue or topic. This value defaults to `com.oracle.webservices.jms.RequestQueue`.
Destination Type	Destination type. Valid values include: `Queue` or `Topic`. This value defaults to `Queue`. This value overrides the `destinationType` value specified as an entry in the Activation Properties field, if applicable. Topics are supported only for one-way communication.
Enable WSDL Access	Boolean flag that specifies whether to publish the WSDL through HTTP. This flag defaults to `true`.
Header Property	JMS header properties. Each property is specified using name-value pairs, separated by semicolons (;). For example: `name1=value1;...;nameN=valueN`.
Message Property	JMS message properties. Each property is specified using name-value pairs, separated by semicolons (;). For example: `name1=value1;...;nameN=valueN`.

Table 21-6 (Cont.) Configuration Properties for SOAP Over JMS Transport

Name	Description
Connection Factory	JNDI name of the connection factory that is used to establish a JMS connection. This value defaults to `com.oracle.webservices.jms.ConnectionFactory`.
Context Parameters	JNDI properties. Each property is specified using name-value pairs, separated by semicolons (;). For example: `name1=value1;...;nameN=valueN`. The properties are added to the `java.util.Hashtable` sent to the `InitialContext` constructor for the JNDI provider.
Context Factory	Name of the initial context factory class for the JNDI lookup. This value maps to the `java.naming.factory.initial` property. This value defaults to `weblogic.jndi.WLInitialContextFactory`.
URL	JNDI provider URL. This value defaults to `t3://localhost:7001`. This value maps to the `java.naming.provider.url` property.
MDB per Destination	Boolean flag that specifies whether to create one listening message-driven bean (MDB) for each requested destination. This value defaults to `true`. If set to `false`, one listening MDB is created for each web service port, and that MDB cannot be shared by other ports.
Message Type	Message type to use with the request message. Valid values are `Bytes` and `Text`. This value defaults to `Bytes`. For more information about configuring the message type, see Configuring the JMS Message Type.
Priority	JMS priority associated with the request and response message. Specify this value as a positive Integer from 0, the lowest priority, to 9, the highest priority. The default value is `0`.
Reply to Name	JNDI name of the JMS destination to which the response message is sent. For a two-way operation, a temporary response queue is generated by default. Using the default temporary response queue minimizes the configuration that is required. However, in the event of a server failure, the response message may be lost. This attribute enables the client to use a previously defined, “permanent" queue or topic rather than use the default temporary queue or topic, for receiving replies. For more information about configuring the JMS response queue, see Configuring the JMS Response Queue. The value maps to the `JMSReplyTo` JMS header in the request message.
Run as Principle Name	Principal used to run the listening MDB.
Run as Role	Role used to run the listening MDB.
Target Service	Port component name of the web service. This value is used by the service implementation to dispatch the service request. If not specified, the service name from the WSDL or `javax.jws.WebService` annotation is used. This value maps to the `SOAPJMS_targetService` JMS message property.
Time to Live	Lifetime, in milliseconds, of the request message. A value of 0 indicates an infinite lifetime. If not specified, the JMS-defined default value (`0`) is used. On the service side, this value also specifies the expiration time for each MDB transaction.

Developing Web Service Clients Using JMS Transport

To develop web service clients using JMS transport, use one of the following methods:

- Update the web service client to enable and configure JMS transport, using one of the following methods:

  - Adding `@com.oracle.webservices.ap.jms.JMSTransportClient` annotation, as described in Using the @JMSTransportClient Annotation.
  - Adding `@com.oracle.webservices.ap.jms.JMSTransportClientFeature` feature client API, as described in Using the JMSTransportClientFeature Client API.
  - Configure the JMS URI as the target endpoint address for synchronous clients, as described in Configuring the JMS URI as the Target Endpoint Address.
- Update the asynchronous web service client to enable and configure JMS transport, as described in Using AsyncClientTransportFeature to Configure Asynchronous Clients.

For more information about these methods, see Invoking a WebLogic Web Service Using JMS Transport.

The above links are in *Developing JAX-WS Web Services for Oracle WebLogic Server*.

How to Use Fast Infoset for Optimizing XML Transmission

Fast Infoset is a compressed binary encoding format that provides a more efficient serialization than the text-based XML format. Fast Infoset optimizes both document size and processing performance.

When enabled, Fast Infoset converts the XML Information Set in the SOAP envelope into a compressed binary format before transmitting the data. Fast Infoset optimizes encrypted and signed messages, MTOM-enabled messages, and SOAP attachments, and supports both HTTP and JMS transports.

The Fast Infoset specification, *ITU-T Rec. X.891 and ISO/IEC 24824-1 (Fast Infoset)* is defined by both the ITU-T and ISO standards bodies. The specification can be downloaded from the ITU Web site: `http://www.itu.int/rec/T-REC-X.891-200505-I/en`

The Fast Infoset capability is enabled on all web services, by default. For web service clients, Fast Infoset is enabled if it is enabled on the web service and advertised in the WSDL.

The following sections describe how to enable and disable Fast Infoset explicitly on web services and clients:

- Configuring Fast Infoset on Web Services
- Configuring Fast Infoset on Web Service Clients

Configuring Fast Infoset on Web Services

The Fast Infoset capability is enabled on all web services, by default.

You can explicitly enable (for example, to ensure the feature is not disabled by a global policy attachment), disable. and configure Fast Infoset for a web service using the following methods:

- Adding the `com.oracle.webservices.api.FastInfosetService` annotation directly in the Java class; the JDeveloper Code Insight feature can help you. For more information, see Configuring Fast Infoset in a Java Class.
- Using the Web Services wizard, as described in Configuring Fast Infoset in the Web Service Wizard.
- Using the Properties window, as described in Configure Fast Infoset in the Properties Window .

Configuring Fast Infoset in a Java Class

You can enable Fast Infoset in a Java class.

To configure Fast Infoset in the Java class:

1. Open the web service class in the source editor.
2. Start typing the annotation, for example, `@FastInfosetService`. 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 explicitly whether the feature is enabled or disabled using the enabled attribute, as shown in the following example.

```
package examples.webservices.hello_world;
import javax.jws.WebService;
import com.oracle.webservices.api.FastInfosetService;

@WebService(name="HelloWorldPortType", serviceName="HelloWorldService")
@FastInfosetService(enabled=true)

public class HelloWorldImpl {
  public String sayHelloWorld(String message) {
  try {
    System.out.println("sayHelloWorld:" + message);
  } catch (Exception ex) { ex.printStackTrace(); }
    return "Message from FI Enabled Service: '" + message + "'";
  }
}
```

Configuring Fast Infoset in the Web Service Wizard

You can enable Fast Infoset 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.

For more information about creating web services using the Create Java Web Service Wizard, see Creating Java Web Services .

To configure Fast Infoset in the web service wizard:

In the Create Java Web Service wizard or web service editor, navigate to the Message Format page and select the **Enable Fast Infoset** checkbox. For more information at any time, press **F1** or click **Help** from within the dialog.

Configure Fast Infoset in the Properties Window

You can enable Fast Infoset in the Properties window.

To configure Fast Infoset in the Properties window:

1. With the web service class open in the source editor, choose **Window > Properties** to open the Properties window.

   For more information at any time, press **F1** or click **Help** from within the Properties window.
2. With the cursor in the public class or `@WebService` line of the class, navigate to the JAX-WS Extensions node in the Properties window.
3. Select **Enable Fast Infoset**.

The `@FastInfosetService` annotation is imported and added to the public class.

Configuring Fast Infoset on Web Service Clients

For web service clients, Fast Infoset is enabled if it is enabled on the web service and advertised in the WSDL.

You can explicitly enable, disable. and configure Fast Infoset for a web service client by passing the `com.oracle.webservices.api.FastInfosetClientFeature` as a parameter when creating the web service proxy or dispatch.For more information about configuring the content negotiation strategy, see "Configuring the Content Negotiation Strategy" in *Developing JAX-WS Web Services for Oracle WebLogic Server*.

The following code excerpt provides an example of using the `com.oracle.webservices.api.FastInfosetClientFeature` feature class to enable and configure Fast Infoset on a web service at design time.

```
package examples.webservices.hello_world.client;

import javax.xml.namespace.QName;
import java.net.MalformedURLException;
import java.net.URL;
import com.oracle.webservices.api.FastInfosetClientFeature;
import com.oracle.webservices.api.FastInfosetContentNegotiationType;

public class Main {

public static void main(String[] args) {
    HelloWorldService service;
    FastInfosetContentNegotiationType clientNeg  =
        FastInfosetContentNegotiationType.PESSIMISTIC;
    FastInfosetClientFeature feature =
FastInfosetClientFeature.builder().fastInfosetContentNegotiation(clientNeg).enabled(true)
.build();
          try {
      service = new HelloWorldService(new URL(args[0] + "?WSDL"), new QName("http://
hello_world.webservices.examples/", "HelloWorldService") );
    } catch (MalformedURLException murl) { throw new RuntimeException(murl); }
      HelloWorldPortType port = service.getHelloWorldPortTypePort(feature);

      String result = null;
      result = port.sayHelloWorld("Hi there!");
      System.out.println( "Got result: " + result );
  }
}
```

To disable Fast Infoset on the client, set the `enabled` flag to `false` or set the content negotiation strategy to `NONE` on the Feature class.

Disabling Fast Infoset on Web Services and Clients

At design time, to disable Fast Infoset explicitly:

- On a web service, set the `enabled` flag to `false` on the annotation. For more information, see Configuring Fast Infoset in a Java Class.
- On a web service client, set the `enabled` flag to `false` or set the content negotiation strategy to `NONE` on the annotation or Feature class. For more information, see Configuring the Content Negotiation Strategy and Example Using FastInfosetClientFeature Class at Design Time in *Developing JAX-WS Web Services for Oracle WebLogic Server*.

The following code excerpt provides an example of using the `com.oracle.webservices.api.FastInfosetService` annotation to disable Fast Infoset on a web service at design time.

```
package examples.webservices.hello_world;
import javax.jws.WebService;
import com.oracle.webservices.api.FastInfosetService;

@WebService(name="HelloWorldPortType", serviceName="HelloWorldService")
@FastInfosetService(enabled=false)

public class HelloWorldImpl {
  public String sayHelloWorld(String message) {
  try {
    System.out.println("sayHelloWorld:" + message);
  } catch (Exception ex) { ex.printStackTrace(); }
    return "Message from FI Enabled Service: '" + message + "'";
  }
}
```

How to Use MTOM for Optimizing Binary Transmission

SOAP Message Transmission Optimization Mechanism/XML-binary Optimized Packaging (MTOM/XOP) defines a method for optimizing the transmission of XML data of type `xs:base64Binary` or `xs:hexBinary` in SOAP messages. When the transport protocol is HTTP, Multipurpose Internet Mail Extension (MIME) attachments are used to carry that data while at the same time allowing both the sender and the receiver direct access to the XML data in the SOAP message without having to be aware that any MIME artifacts were used to marshal the `base64Binary` or `hexBinary` data.

The binary data optimization process involves the following steps:

1. Encode the binary data.
2. Remove the binary data from the SOAP envelope.
3. Compress the binary data.
4. Attach the binary data to the MIME package.
5. Add references to the MIME package in the SOAP envelope.

MTOM/XOP support is standard in JAX-WS via the use of JWS annotations. The MTOM specification does not require that, when MTOM is enabled, the Web service runtime use XOP binary optimization when transmitting `base64binary` or `hexBinary` data. Rather, the specification allows the runtime to choose to do so. This is because in certain cases the runtime may decide that it is more efficient to send the binary data directly in the SOAP Message; an example of such a case is when transporting small amounts of data in which the overhead of conversion and transport consumes more resources than just inlining the data as is.

The following Java types are mapped to the `base64Binary` XML data type, by default: `javax.activation.DataHandler`, `java.awt.Image`, and `javax.xml.transform.Source`. The elements of type `base64Binary` or `hexBinary` are mapped to `byte[]`, by default.

The following sections describe how to enable MTOM on web services and clients:

- Enabling MTOM on Web Services
- Enabling MTOM on Web Service Clients
- Configuring MTOM on Web Services and Clients

Enabling MTOM on Web Services

You can enable MTOM for a web service using the following

- Adding the `javax.xml.ws.soap.MTOM` annotation directly in the Java class; the JDeveloper Code Insight feature can help you. For more information, see Configuring Fast Infoset in a Java Class.
- Using the Web Services wizard, as described in Configuring Fast Infoset in the Web Service Wizard.
- Using the Properties window, as described in Configure Fast Infoset in the Properties Window .

Enabling MTOM in a Java Class

You can enable MTOM in a Java class.

To enable MTOM in the Java class:

1. Open the web service class in the source editor.
2. Start typing the annotation, for example, `@MTOM`. When you pause, or click Ctrl+Shift+Space, a popup appears from which you can choose the correct entry to complete the statement.

The following provides an example of enabling MTOM on a web service:

```
package examples.webservices.mtom;

import javax.jws.WebMethod;
import javax.jws.WebService;
import javax.xml.ws.soap.MTOM;

@MTOM
@WebService(name="MtomPortType",
            serviceName="MtomService",
            targetNamespace="http://example.org")
public class MTOMImpl {
  @WebMethod
  public String echoBinaryAsString(byte[] bytes) {
    return new String(bytes);

  }
}
```

Enabling MTOM by Attaching a WebLogic Web Service Policy

You can enable MTOM by attaching a WebLogic web service policy, as described in "Enabling MTOM on the Web Services by Attaching a WS-Policy File" in *Developing JAX-WS Web Services for Oracle WebLogic Server*.

Enabling MTOM in the Web Service Wizard

You can enable MTOM 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.

For more information about creating web services using the Create Java Web Service Wizard, see Creating Java Web Services .

To enable MTOM in the web service wizard, in the Create Java Web Service wizard or web service editor, navigate to the Message Format page and select the **Enable MTOM** checkbox. For more information at any time, press **F1** or click **Help** from within the dialog.

Enabling MTOM in the Properties Window

You can enable MTOM in the Properties window.

To enable MTOM in the Properties window:

1. With the web service class open in the source editor, choose **Window > Properties** to open the Properties window.

   For more information at any time, press **F1** or click **Help** from within the Properties window.
2. With the cursor in the public class or `@WebService` line of the class, navigate to the JAX-WS node in the Properties window.
3. Select **Enable MTOM**.

The `@FastInfosetService` annotation is imported and added to the public class.

Enabling MTOM on Web Service Clients

To enable MTOM on the client of the Web service, pass an instance of the `javax.xml.ws.soap.MTOMFeature` as a parameter when creating the Web service proxy or dispatch, as illustrated in the following example. Relevant code is shown in **bold**.

```
package examples.webservices.mtom.client;

import javax.xml.ws.soap.MTOMFeature;

public class Main {
  public static void main(String[] args) {
    String FOO = "FOO";
    MtomService service = new MtomService()
    MtomPortType port = service.getMtomPortTypePort(new MTOMFeature());
    String result = null;
    result = port.echoBinaryAsString(FOO.getBytes());
    System.out.println( "Got result: " + result );
  }
}
```

Configuring MTOM on Web Services and Clients

At design time, you can configure the following

- Set an attachment threshold to specify when the `xs:binary64` data is sent inline or as an attachment. By default, the attachment threshold is 0 bytes. For more information, see Setting an Attachment Threshold.
- Enable HTTP chunking on the transport layer to minimize excessive buffering on the client side when processing MTOM attachments. For more information, see Enabling HTTP Chunking.

How to Manage WSDL Files

JDeveloper provides a number of ways that you can manage WSDL files for a web service, as described in the following sections:

- Creating WSDL Documents
- Displaying the WSDL for a Web Service
- Adding a WSDL to a Web Service Project
- Saving a WSDL to Your Local Directory
- Viewing the WSDL Used to Create the Web Service Client
- Refreshing the Local Copy of the WSDL and Regenerating the Web Service Client Proxy and Classes
- Updating the Web Service WSDL Used by the Client at Run Time

Creating WSDL Documents

You can create a WSDL document, for example, to create a top-down web service.

To create a WSDL document:

1. In the Applications window, select the project containing the Java class or EJB from which you want to create a web service.
2. Choose **File > New > From Gallery** 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.

Displaying 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.

Update “Managing WSDL Files" to indicate that when a user views a locally saved WADL file, the original WSDL location is saved as a read-only field in the editor.

To display the WSDL to a web service project

In the Applications window, 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.

Adding 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")
```

To add a WSDL to a web service project

In the Applications window, 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.

> **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.

Saving 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.

Viewing the WSDL Used to Create the Web Service Client

You can view the WSDL that was used to generate the web service client. Please note:

- 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 Creating 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 Regenerating Web Service Client and Proxy Classes.

- If the local version is not available, the remote WSDL is displayed.

To view the WSDL used to create the web service client:

1. Right-click on the web service client container within the Applications window.
2. Select **Go To WSDL** from the pop-up menu.

The WSDL is displayed.

Refreshing the Local Copy of the WSDL and Regenerating the Web Service Client Proxy and Classes

You can refresh the local copy of the WSDL from the original WSDL location. The web service client and proxy classes are regenerated once the WSDL is refreshed.

To refresh the local copy of the WSDL:

1. In the Applications window, 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.

Updating the Web Service WSDL Used by the Client 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

Using 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 https://www.oasis-open.org/committees/entity/spec.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`

Using 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: How to Use 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: How to Update 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 *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 `CreditRatingService` 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>
```

How to Edit JAX-WS Web Services

You can edit a JAX-WS web service that you have created in JDeveloper, for example to change the exposed method or a file location.

When you edit a JAX-WS 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.

To edit a JAX-WS web service:

1. In the Applications window, right-click the web service and choose **Web Service 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 Deploying Web Services.

How to Delete JAX-WS Web Services

You can delete a JAX-WS web service if it is no longer needed.

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.

To delete a web service

In the Applications window, right-click the web service and choose **Delete**. The Confirm Delete dialog displays listing the file usages. Click **OK**.

Creating RESTful Web Services and Clients

REpresentational State Transfer (REST) is a key design idiom that embraces a stateless client-server architecture in which the web services are viewed as resources and can be identified by their URLs. Learn how to create web services and clients using JDeveloper.

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`.

JAX-RS is a Java programming language API that uses annotations to simplify the development of RESTful web services. JAX-RS annotations are runtime annotations. When you deploy the Java EE application archive containing JAX-RS resource classes to WebLogic Server, the runtime configures the resources, generates the helper classes and artifacts, and exposes the resource to clients.

WebLogic Server supports the following JAX-RS Reference Implementations (RIs):

- **Jersey 1.18 (JAX-RS 1.1 RI)**—This is the default implementation and provides a production quality implementation of the JSR-311 JAX-RS 1.1 specification, defined at: `http://jcp.org/en/jsr/summary?id=311`.

> **Note:**
>
> Although support for the Jersey 1.18 (JAX-RS 1.1RI) client APIs of WebLogic Server is deprecated, support is maintained for backward compatibility. For information about compatibility with earlier Jersey/JAX-RS releases, see *Compatibility with Earlier Jersey/JAX-RS Releases*.

- **Jersey 2.5.1 (JAX-RS 2.0 RI)**—This implementation is offered as a shared library and provides a production quality implementation of the JSR-339 JAX-RS 2.0 specification, defined at: `http://jcp.org/en/jsr/detail?id=339`.

The Jersey 2.5.1 (JAX-RS 2.0 RI) shared library is auto-deployed to the Integrated WebLogic Server.

Note: You can attach OWSM policies to RESTful web services and clients that are built using Jersey 1. *x* JAX-RS RI. RESTful web services and clients that are built using Jersey 2.5.1 (JAX-RS 2.0 RI) are also secured using OWSM policies. For information about securing RESTful web services and clients built using Jersey 2.5.1 (JAX-RS 2.0 RI), see "Securing RESTful Web Services and Clients".

For complete details about developing RESTful web services and clients using JAX-RS, see:

- *Developing and Securing RESTful Web Services for Oracle WebLogic Server*
- Jersey RI documentation at: `https://wikis.oracle.com/display/Jersey/Main`

The following sections describe how to create RESTful web service and clients quickly and easily using JDeveloper:

- How to Create RESTful Web Services
- How to Create RESTful Web Service Clients

How to Create RESTful Web Services

You can develop RESTful web services quickly and easily using JDeveloper. All of the standard Java source editor features will work with RESTful web service calls, such as code insight, import assistance, and so on.

The following sections describe how to develop RESTful web services:

- "Example of a Simple RESTful Web Service"
- "Creating a RESTful Web Service"
- "Defining the Relative URI of the Root Resource and Subresources"
- "Mapping Incoming HTTP Requests to Java Methods"
- "Customizing Media Types for the Request and Response Messages"
- "Extracting Information from the Request Message"
- "Mapping HTTP Request and Response Entity Bodies Using Entity Providers"
- "Accessing the RESTful Web Service WADL"

For information about:

- Developing RESTful web services, see Securing RESTful Web Services and Clients in *Developing and Securing RESTful Web Services for Oracle WebLogic Server*.
- Administering RESTful web services, see *Administering Web Services*.

Example of a Simple RESTful Web Service

The following is an example of a simple RESTful web service. In this example:

- The `helloWorld` class is a resource with a relative URI path defined as `/helloworld`. At runtime, if the context root for the WAR file is defined as `http://examples.com`, the full URI to access the resource is `http://examples.com/helloworld`. For more information, see Defining the Relative URI of the Root Resource and Subresources.
- The `sayHello` method supports the HTTP GET method. For more information, see Mapping Incoming HTTP Requests to Java Methods.
- The `sayHello` method produces content of the MIME media type `text/plain`. For more information, see Customizing Media Types for the Request and Response Messages.

Additional examples are listed in Learn More About RESTful Web Services in *Developing and Securing RESTful Web Services for Oracle WebLogic Server*.

```
package samples.helloworld;

import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;

// Specifies the path to the RESTful web service
@Path("/helloworld")
public class helloWorld {

   // Specifies that the method processes HTTP GET requests
   @GET
   @Path("hello")
   @Produces("text/plain")
   public String sayHello() {
      return "Hello World!";
   }
}
```

Creating a RESTful Web Service

You can create a new RESTful web service class or generate a RESTful web service from an existing Java class using the Create RESTful Service wizard.

The wizard creates the deployment files for you, so once you create your web service the final step is to deploy it. You can test them using the HTTP Analyzer. For more information, see How to Examine Web Services using the HTTP Analyzer .

As described in How to Choose Your Deployment Platform, when creating a RESTful web service using the Create RESTful Service wizard, you are prompted to select the deployment platform and you can choose the Jersey 1.0 or 2.0 style.

The JAX-RS Jersey library is automatically added to your project. If required by your application, you can add to your project the JAX-RS Jersey Jackson or JAX-RS Jersey Jettison libraries, which are bundled with the product. (The JAX-RS Jersey Rome library is not bundled with the product.) For information about adding a library to your project, see "How to Manage Libraries".

> **Note:**
>
> As support for the server-side Jersey 1.x APIs is unavailable, use the corresponding JAX-RS 2.0 standard. If required, use Jersey 2.x APIs. Support for the Jersey 1.18 (JAX-RS 1.1RI) client APIs of WebLogic Server is deprecated. However, support is maintained for backward compatibility. For more information about compatibility with earlier Jersey/JAX-RS releases see, *Developing and Securing RESTful Web Services for Oracle WebLogic Server*.

The Jersey 2.5.1 (JAX-RS 2.0 RI) shared library is auto-deployed to the Integrated WebLogic Server.

> **Note:**
>
> You can attach OWSM policies to RESTful web services and clients that are built using Jersey 1.18 (JAX-RS 1.1 RI). RESTful web services and clients that are built using Jersey 2.5.1 (JAX-RS 2.0 RI) are secured using OWSM policies. For more information about securing RESTful web services and clients built using Jersey 2.5.1 (JAX-RS 2.0 RI), see Securing RESTful Web Services and Clients.

To create the RESTful web service:

1. In the Applications window, select the project within which you want to create a new RESTful web service or that contains the Java class from which you want to create a RESTful web service or within which you want to create a new RESTful web service.
2. Choose **File > New > From Gallery** to open the New Gallery.
3. In the **Categories** list, expand **Business Tier** and select **Web Services**.
4. In the **Items** list, select **RESTful Service** and click **OK** to launch the Create RESTful Service wizard.

   For detailed help about completing the wizard, press **F1** or click **Help** from within the wizard.

Alternatively, you can launch the Create RESTful Service wizard by right-clicking on the Java class from which you want to create a RESTful web service and selecting **Create RESTful Service**.

Defining the Relative URI of the Root Resource and Subresources

Add the `javax.ws.rs.Path` annotation at the class level of the resource to define the relative URI of the RESTful web service. Such classes are referred to a **root resource classes**. For more information about the root resource class, see the following sections in *Developing and Securing RESTful Web Services for Oracle WebLogic Server*:

- Defining the Root Resource Class
- How to Define the Relative URI of the Resource Class (@Path)

You can add `@Path` on methods of the root resource class as well, to define subresources to group specific functionality. For more information, see How to Define the Relative URI of Subresources (@Path).

You can define the URI as a constant or variable value (referred to as a “URI path template"):

- To define the URI as a constant value, pass a constant value to the `@Path` annotation. Preceding and ending slashes (/) are optional.
- To define the URI as a URI path template, pass one or more variable values enclosed in braces in the `@Path` annotation. Then, you can use the `javax.ws.rs.PathParam` annotation to extract variable information from the request URI, defined by the `@Path` annotation, and initialize the value of the method parameter, as described in Extracting Information from the Request Message.

For more information, see How to Define the Relative URI of the Resource Class (@Path).

Using JDeveloper, you can specify the `@Path` annotation using one of the following methods:

- When creating the RESTful web service using the Create RESTful Service wizard, as described in Defining the @Path Annotation in the RESTful Service Wizard.
- In the Java class, as described in Defining the @Path Annotation in the Java Class.
- In the Properties window, as described in Defining the @Path Annotation in the Properties Window.

Defining the @Path Annotation in the RESTful Service Wizard

You can define the relative URI of the root resource and subresources when creating your RESTful service using the Create RESTful Service wizard. For more information about invoking the wizard, see Creating a RESTful Web Service.

To define the @Path annotation in the RESTful service wizard:

In the Create RESTful Service wizard, navigate to one of the following pages, depending on whether you are creating a new RESTful service class or generating one from an existing Java class:

- Create New RESTful Service
- Create RESTful Service From Java Class

Define the relative URIs, as follows:

- To define the relative URI for the root resource class, enter a value in the **Root Path** field.
- To define the relative URI for the subresource, enter a value in the **Path** field of the HTTP Methods table.

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

Defining the @Path Annotation in the Java Class

You can define the relative URI of the root resource and subresources when creating your RESTful service by adding the `@Path` annotation directly in the Java class; the Code Insight feature can help you. For more information, see How to Work with Web Services Code Insight.

To define the @Path annotation in the Java class:

1. Open the RESTful service class in the source editor.
2. Perform one or more of the following tasks:
   - Add the `@Path` annotation at the class level of the resource to define the relative URI of the RESTful service.
   - Add the `@Path` annotation to the method of a resource to define a subresource.

You can use the Code Insight to help you. Start typing the annotation, for example, `@Path`. When you pause, or click Ctrl+Shift+Space, a popup appears from which you can choose the correct entry to complete the statement.

> **Note:**
>
> Ensure that you select `javax.ws.rs.Path` from the popup. For example, do not select `java.nio.file.Path` inadvertently.

For more information about defining the `@Path` annotation, see Defining the Relative URI of the Root Resource and Subresources in *Developing and Securing RESTful Web Services for Oracle WebLogic Server*.

Defining the @Path Annotation in the Properties Window

You can define the relative URI of the root resource and subresources using the Properties window.

To define the @Path annotation in the Properties window:

1. With the RESTful service class open in the source editor, choose **Window > Properties** to open the Properties window.

   For more information at any time, press **F1** or click **Help** from within the Properties window.
2. To define the relative URI of the resource class:
   a. Position your cursor at the class level of the resource.
   b. In the Properties window, expand JAX-RS and click **RESTful Resource Class**.

      The service is updated to add the `@Path` annotation above the resource class and import `javax.ws.rs.Path`.
   c. Enter a value in the **Path** field to define the URI. The URI can be defined as a constant value or URI path template.

   For more information, see How to Define the Relative URI of the Resource Class (@Path) in *Developing and Securing RESTful Web Services for Oracle WebLogic Server*.
3. To define the relative URI of a subresource:
   a. Position your cursor at the method of the resource.
   b. If you have not already mapped the resource method to an HTTP method, in the Properties window, expand JAX-RS and select an HTTP method from the **Method Type** drop-down menu. Valid HTTP methods include: `GET`, `POST`, `PUT`, or `DELETE`.

      The service is updated to include the method annotation above the resource method and the appropriate API is imported. For more information, see Mapping Incoming HTTP Requests to Java Methods.
   c. Enter a value in the **Method Path** field to define the relative URI of the subresource. The URI can be defined as a constant value or URI path template.

   For more information, see How to Define the Relative URI of Subresources (@Path).

What Happens at Runtime: How the Base URI is Constructed

The base URI of the RESTful web service is constructed as follows:

`http://myHostName/contextPath/servletURI/resourceURI`

For a complete description of the base URI path structure, see What Happens at Runtime: How the Base URI is Constructed in *Developing and Securing RESTful Web Services for Oracle WebLogic Server*.

For the `contextPath`, or context root, at the project level you can set the value that will be assigned to the deployed Java EE web application on Integrated WebLogic Server. For more information, see How to Set the Context Root for Web Services. In this scenario, the `contextPath` defaults to:

applicationname-projectname`-context-root`

Mapping Incoming HTTP Requests to Java Methods

JAX-RS uses Java annotations to map an incoming HTTP request to a Java method. Table 21-7 lists the annotations available, which map to the similarly named HTTP methods.

Table 21-7 javax.ws.rs Annotations for Mapping HTTP Requests to Java Methods

Annotation	Description	Idempotent
`@GET`	Transmits a representation of the resource identified by the URI to the client. The format might be HTML, plain text, JPEG, and so on. For more information, see How to Transmit a Representation of the Resource (@GET).	Yes
`@PUT`	Creates or updates the representation of the specified resource identified by the URI. For more information, see How to Create or Update the Representation of the Resource (@PUT).	Yes
`@DELETE`	Deletes the representation of the resource identified by the URI. For more information, see How to Delete a Representation of the Resource (@DELETE).	Yes
`@POST`	Creates, updates, or performs an action on the representation of the specified resource identified by the URI. For more information, see How to Create, Update, or Perform an Action on a Representation of the Resource (@POST).	No
`@HEAD`	Returns the response headers only, and not the actual resource (that is, no message body). This is useful to save bandwidth to check characteristics of a resource without actually downloading it. For more information, see `http://docs.oracle.com/javaee/6/api/index.html?javax/ws/rs/HEAD.html`. The `HEAD` method is implemented automatically if not implemented explicitly. In this case, the runtime invokes the implemented `GET` method, if present, and ignores the response entity, if set.	Yes
`@OPTIONS`	Returns the communication options that are available on the request/response chain for the specified resource identified by the URI. The `Allow` response header will be set to the set of HTTP methods supported by the resource and the WADL file is returned. For more information, see `http://docs.oracle.com/javaee/6/api/index.html?javax/ws/rs/OPTIONS.html`. The `OPTIONS` method is implemented automatically if not implemented explicitly. In this case, the `Allow` response header is set to the set of HTTP methods supported by the resource and the WADL describing the resource is returned.	Yes
`@HttpMethod`	Indicates that the annotated method should be used to handle HTTP requests. For more information, see `http://docs.oracle.com/javaee/6/api/index.html?javax/ws/rs/HttpMethod.html`.	N/A

Using JDeveloper, you can map HTTP requests to Java methods using one of the following methods:

- When creating the RESTful web service using the Create RESTful Service wizard, as described in Mapping HTTP Requests to Java Methods in the RESTful Service Wizard.
- In the Java class, as described in Mapping HTTP Requests to Java Methods in the Java Class.
- In the Properties window, as described in Mapping HTTP Requests to Java Methods in the Properties Window.

Mapping HTTP Requests to Java Methods in the RESTful Service Wizard

You can map HTTP requests to Java methods when creating your RESTful web service using the Create RESTful Service wizard. For more information about invoking the wizard, see Creating a RESTful Web Service.

To map HTTP requests to Java methods in the RESTful service wizard:

In the Create RESTful Service wizard, navigate to one of the following pages, depending on whether you are creating a new RESTful service class or generating one from an existing Java class:

- Create New RESTful Service
- Create RESTful Service From Java Class

Perform one of the following tasks:

- If creating a new RESTful service class, select the HTTP method(s) for which you want to create Java methods. In each case, enter a value in the Nos column to indicate the desired number of Java methods that you want to be mapped to HTTP methods.
- If generating a RESTful service from an existing class, map HTTP requests to Java methods in the Configure HTTP Methods table by selecting an HTTP method from the **Type** drop-down list. Valid HTTP methods include: `GET`, `POST`, `PUT`, or `DELETE`.

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

Mapping HTTP Requests to Java Methods in the Java Class

You can map HTTP requests to Java methods when creating your RESTful service by adding one of the HTTP request annotations defined in Table 21-7 directly in the Java class; the Code Insight feature can help you. For more information, see How to Work with Web Services Code Insight.

To map HTTP requests to Java methods in the Java class:

1. Open the RESTful service class in the source editor.
2. Add an HTTP request annotation from Table 21-7 to one or more methods in the Java class.

   You can use the Code Insight to help you. Start typing the annotation, for example, `@GET`. 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 defining the HTTP request annotation, see Mapping Incoming HTTP Requests to Java Methods in *Developing and Securing RESTful Web Services for Oracle WebLogic Server*.

Mapping HTTP Requests to Java Methods in the Properties Window

You can map HTTP requests to Java methods in the Properties window.

To map HTTP requests to Java methods in the Properties window:

1. With the RESTful service class open in the source editor, choose **Window > Properties** to open the Properties window.

   For more information at any time, press **F1** or click **Help** from within the Properties window.

2. Position your cursor at the method of the resource.

3. In the Properties window, expand JAX-RS and select an HTTP method from the **Method Type** drop-down menu. Valid HTTP methods include: `GET`, `POST`, `PUT`, or `DELETE`.

   The service is updated to include the method annotation above the resource method and the appropriate API is imported.

For more information about defining the HTTP request annotation, see Mapping Incoming HTTP Requests to Java Methods in *Developing and Securing RESTful Web Services for Oracle WebLogic Server*.

Customizing Media Types for the Request and Response Messages

Add the `javax.ws.rs.Consumes` or `javax.ws.rs.Produces` annotation at the class or method level of the resource to customize the media type of the request and response messages, respectively. More than one media type can be declared in each case.

Using JDeveloper, you can customize messages types for the request and response messages using one of the following methods:

- When creating the RESTful web service using the Create RESTful Service wizard, as described in Customizing Media Types in the RESTful Service Wizard.
- In the Java class, as described in Customizing Media Types in the Java Class.
- In the Properties window, as described in Customizing Media Types in the Properties Window.

Customizing Media Types in the RESTful Service Wizard

You can customize media types when creating your RESTful web service using the Create RESTful Service wizard. For more information about invoking the wizard, see Creating a RESTful Web Service.

To customize media types in the RESTful service wizard:

In the Create RESTful Service wizard, navigate to one of the following pages, depending on whether you are creating a new RESTful service class or generating one from an existing Java class:

- Create New RESTful Service
- Create RESTful Service From Java Class

Customize the media types, as follows:

- To customize media types for the root resource class, click **...** next to the **Consumes** or **Produces** field, select one or more media types from the Select Media Types dialog box, and click **OK**. The number of media types configured is reflected in the field.
- To customize media types for the methods:
  - If creating a new RESTful service class, in the Select HTTP Methods table click in the **Consumes** or **Produces** column corresponding to the method for which you want to

configure media types, select the media types from the Select Media Types dialog box, and click **OK**. The number of media types configured is reflected in the table entry.

- If generating a RESTful service from an existing class, in the Configure HTTP Methods table click in the **Consumes** or **Produces** column corresponding to the method for which you want to configure media types, select the media types from the Select Media Types dialog box, and click **OK**. The number of media types configured is reflected in the table entry.

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

Customizing Media Types in the Java Class

You can customize media types when creating your RESTful service by adding the `@Produces` or `@Consumes` annotation at the class or method level of the resource to customize the media types of the request and response messages, respectively, directly in the Java class; the Code Insight feature can help you. For more information, see How to Work with Web Services Code Insight.

To customize media types in the Java class:

1. Open the RESTful service class in the source editor.
2. Add the `@Produces` or `@Consumes` annotation at the class or method level.

   You can use the Code Insight to help you. Start typing the annotation, for example, `@Produces`. 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 customizing media types, see Customizing Media Types for the Request and Response Messages in *Developing and Securing RESTful Web Services for Oracle WebLogic Server*.

Customizing Media Types in the Properties Window

You can customize media types in the Properties window.

To customize media types in the Properties window:

1. With the RESTful service class open in the source editor, choose **Window > Properties** to open the Properties window.

   For more information at any time, press **F1** or click **Help** from within the Properties window.
2. Position your cursor at the class or method level of the resource.
3. In the Properties window, expand JAX-RS, click **...** next to the **Consumes Type** or **Produces Type** field, select one or more media types from the Edit Property dialog box, and click **OK**. The number of media types configured is reflected in the field.

   The service is updated to include the method annotation and the appropriate API is imported.

For more information about customizing media types, see Customizing Media Types for the Request and Response Messages in *Developing and Securing RESTful Web Services for Oracle WebLogic Server*.

Extracting Information from the Request Message

The `javax.ws.rs` package defines a set of annotations, shown in Table 21-8, that enable you extract information from the request message to inject into parameters of your Java method.

Table 21-8 javax.ws.rs Annotations for Extracting Information From the Request Message

Annotation	Description
`@CookieParam`	Extract information from the HTTP cookie-related headers to initialize the value of a method parameter. For more information, see `http://docs.oracle.com/javaee/6/api/index.html?javax/ws/rs/CookieParam.html`.
`@DefaultValue`	Define the default value of the request metadata that is bound using one of the following annotations: `@CookieParam`, `@FormParam`, `@HeaderParam`, `@MatrixParam`, `@PathParam`, or `@QueryParam`. For more information, see How to Define the DefaultValue (@DefaultValue) in *Developing and Securing RESTful Web Services for Oracle WebLogic Server*.
`@FormParam`	Extract information from an HTML form of the type `application/x-www-form-urlencoded`. For more information, see `http://docs.oracle.com/javaee/6/api/index.html?javax/ws/rs/FormParam.html`.
`@HeaderParam`	Extract information from the HTTP headers to initialize the value of a method parameter. For more information, see `http://docs.oracle.com/javaee/6/api/index.html?javax/ws/rs/HeaderParam.html`.
`@MatrixParam`	Extract information from the URI path segments to initialize the value of a method parameter. For more information, see `http://docs.oracle.com/javaee/6/api/index.html?javax/ws/rs/MatrixParam.html`.
`@PathParam`	Define the relative URI as a variable value (referred to as “URI path template"). For more information, see How to Extract Variable Information from the Request URI (@PathParam).
`@QueryParam`	Extract information from the query portion of the request URI to initialize the value of a method parameter. For more information, see How to Extract Request Parameters (@QueryParam).

Using JDeveloper, you can extract information from the request message using one of the following methods:

- When creating the RESTful web service using the Create RESTful Service wizard, as described in Extracting Information from the Request Message in the RESTful Service Wizard.
- In the Java class, as described in Extracting Information from the Request Message in the Java Class.

In the Properties window, you can enable encoding of a parameter value that is bound using one of the following annotations: `@FormParam`, `@MatrixParam`, `@PathParam`, or `@QueryParam`. For more information, see Enabling the Encoding Parameter Values in the Properties Window.

Extracting Information from the Request Message in the RESTful Service Wizard

You can extract information from the request when creating your RESTful web service using the Create RESTful Service wizard. For more information about invoking the wizard, see Creating a RESTful Web Service.

To extract information from the request message in the RESTful service wizard:

When using the RESTful service wizard, you can extract information from the request message when generating a RESTful service from an existing Java class only.

1. In the Create RESTful Service wizard, navigate to the Create RESTful Service From Java Class page.
2. Select a method in the Configure HTTP Methods table. Ensure that you have selected an HTTP method from the Type column for the method.

   A list of the method parameters is shown in the Configure Parameters table.

3. In the Configure Parameters table:
   - In the **Annotation** drop-down list, select the annotation to use to extract information from the request message. A list of valid annotations is defined in Table 21-8.
   - In the **Parameter** field, enter the name of the parameter that will be used to store the extracted value.
   - In the **Default** field, enter the default value for the parameter, if no value is passed with the request message.
   - Click **Encode** to enable encoding of the parameter value. This field is enabled for the following annotations only: `@FormParam`, `@MatrixParam`, `@PathParam`, or `@QueryParam`.
4. Click **Next** to advanced to the next screen or **Finish** to generate the RESTful service.

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

Extracting Information from the Request Message in the Java Class

You can extract information from the request message when creating your RESTful service by adding one or more of the annotations defined in Table 21-8 to the class or method level of the resource directly in the Java class; the Code Insight feature can help you. For more information, see How to Work with Web Services Code Insight.

To extract information from the request message in the Java class:

1. Open the RESTful service class in the source editor.
2. Add one of the annotations from Table 21-8 at the class or method level.

   You can use the Code Insight to help you. Start typing the annotation, for example, `@PathParam`. When you pause, or click Ctrl+Shift+Space, a popup appears from which you can choose the correct entry to complete the statement.
3. Add the `javax.ws.rs.Encode` annotation at the class or method level to enable encoding of a parameter value that is bound using one of the following annotations: `@FormParam`, `@MatrixParam`, `@PathParam`, or `@QueryParam`. If specified at the class level, parameters for all methods in the class will be encoded. For more information, see Enabling the Encoding Parameter Values (@Encode).

For more information about extracting information from the request message, see Extracting Information From the Request Message.

Enabling the Encoding Parameter Values in the Properties Window

In the Properties window, you can enable the encoding of parameter values that are bound using one of the following annotations: `@FormParam`, `@MatrixParam`, `@PathParam`, or `@QueryParam`. For more information about the `@Encode` annotation, see Encoding Parameter Values (@Encode) in *Developing and Securing RESTful Web Services for Oracle WebLogic Server*.

To enable the encoding of parameter values in the Properties window:

1. With the RESTful service class open in the source editor, choose **Window > Properties** to open the Properties window.

   For more information at any time, press **F1** or click **Help** from within the Properties window.
2. Position your cursor at the class or method level of the resource.
3. In the Properties window, expand JAX-RS and click **Encode Values**.

The service is updated to include the `@Encode` annotation and import the `javax.ws.rs.Encoded` API.

Note: If you enable encoding at the class level, encoding is enabled on all associated methods, and the Encode Values checkbox is not available at the method level.

Mapping HTTP Request and Response Entity Bodies Using Entity Providers

A subset of Java types are supported automatically by HTTP request and response entity bodies. For return types that are not supported automatically, you must define an entity provider to map HTTP request and response entity bodies to method parameters and return types. For a list of supported Java types and more information about creating an entity provider, see “Mapping HTTP Request and Response Entity Bodies Using Entity Providers" in *Developing and Securing RESTful Web Services for Oracle WebLogic Server*.

When creating a RESTful web service from an existing Java class using the Create RESTful Service wizard (by right-clicking on the Java class and selecting **Create RESTful Service**), if the class contains methods that utilize types that are not supported automatically, the following warning message is displayed:

```
The return types for the following methods are not supported automatically for
HTTP response entities and require entity providers to map between
representations and Java types. Check and correct the code after the service is
generated.
methodName
```

The following code excerpt provides an example of a class that contains a method (`getClass`) that returns a custom type, and that requires you to write an entity provider.

```
public class Class1
{
  public String hello() { return "Hello"; }
  public Class2 getClass(String name) { return new Class2(); };
}

public class Class2
{
  public Class2() { }
}
```

> **Note:**
>
> Jersey JSON provides a set of JAX-RS `MessageBodyReader` and `MessageBodyWriter` providers distributed with the jersey-json module. For more information, see “JSON Support" in the `Jersey User Guide` at: `https://eclipse-ee4j.github.io/jersey.github.io/documentation/2.35/user-guide.html`
>
> If required by your application, you can add to your project the JAX-RS Jersey Jackson, which is bundled with the product. For information about adding a library to your project, see How to Manage Libraries.

Accessing the RESTful Web Service WADL

The Web Application Description Language (WADL) is an XML-based file format that describes your RESTful web services application. By default, a basic WADL is generated at runtime and can be accessed from JDeveloper.

To access the RESTful web service WADL:

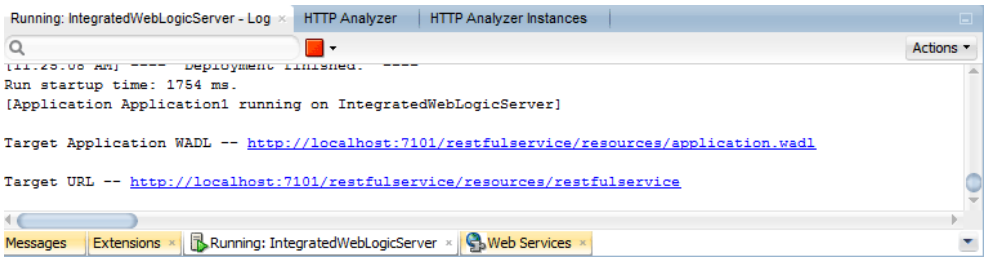
1. Right-click the service in the Applications window and choose **Test Web Service**.

   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 Servers window.
   - Displays a Log window for the integrated application server (if the log window is not already open).

2. Click the **Target Application WADL** provided in the integrated application server log window. For example:

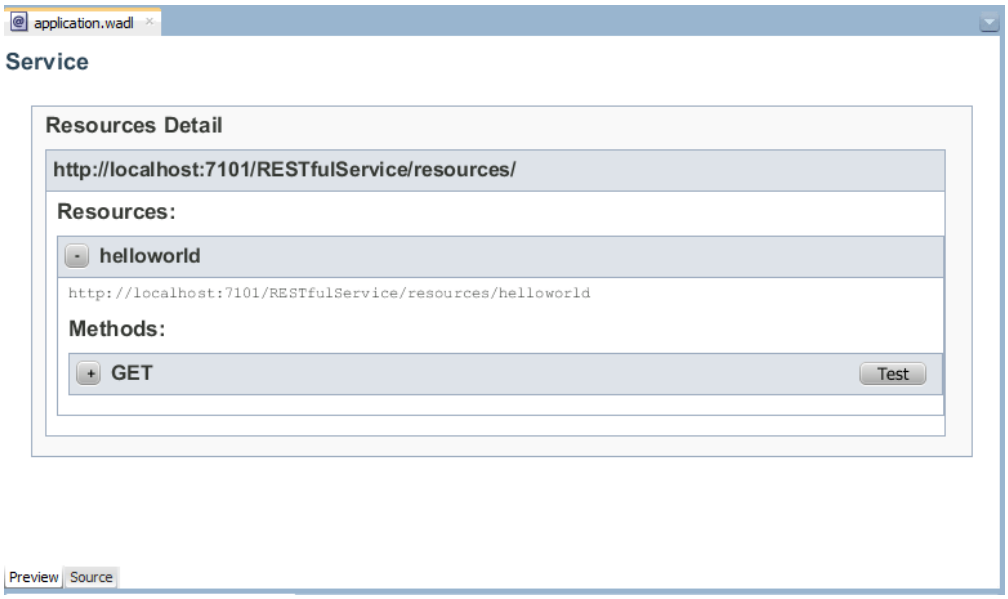
Figure 21-1 Target Application WADL Link



> **Note:**
>
> You can view the WADL from the HTTP Analyzer window by clicking the WADL URI in the HTTP Analyzer window.

A read-only version of the WADL file opens in the source editor, as shown in the figure.

Figure 21-2 Example of WADL File



The figure shows the WADL file structure, including the resources, subresources, and methods in the RESTful web service. You can press Test to invoke the HTTP Analyzer.

You can toggle between the following views:

- Preview—Enables you to:
  - View a summary of resources and subresources in the RESTful web service.
  - View the methods mapped for each resource, and the request and response representations and status.
  - Test each method by clicking the **Test** button to invoke the HTTP Analyzer.
- Source—View the source of the WADL.

How to Create RESTful Web Service Clients

JDeveloper makes it easy to use a RESTful web service in your application by allowing you to create client and proxy classes to access the service using the Create RESTful Proxy Client wizard. You can launch the wizard when you locate or create a RESTful web service. Alternatively, you can launch the wizard directly and enter the URL for the RESTful web service or use the Find Web Service wizard to locate a RESTful web service in a UDDI registry.

As described in How to Choose Your Deployment Platform, when creating a RESTful web service client using the Create RESTful Proxy Client wizard, you are prompted to select the deployment platform, and you can choose the Jersey 1.18 (JAX-RS 1.1) or Jersey 2.5 (JAX-RS 2.0 RI).

The Jersey JAX-RS library is automatically added to your project. If required by your application, you can add to your project the Jersey JAX-RS Jackson or Jersey JAX-RS Jettison libraries, which are bundled with the product. (The Jersey JAX-RS Rome library is not bundled with the product.) For information about adding a library to your project, see How to Manage Libraries.

The Jersey 2.5.1 (JAX-RS 2.0 RI) shared library is auto-deployed to the Integrated WebLogic Server.

Please note:

- For Jersey 1.18 (JAX-RS 1.1 RI), the RESTful web service client API is provided by the Jersey JAX-RS RI specifically; they are not part of the JAX-RS standard. Although support for the Jersey 1.18 (JAX-RS 1.1RI) client APIs of WebLogic Server is deprecated, support is maintained for backward compatibility. For more information about compatibility with earlier Jersey/JAX-RS releases, see *Compatibility with Earlier Jersey/JAX-RS Releases*.

  For Jersey 2.5.1 (JAX-RS 2.0 RI), a standard client API is supported.
- You can attach OWSM policies to RESTful web services and clients that are built using Jersey 1.18 (JAX-RS 1.1 RI) . RESTful web services and clients that are built using Jersey 2.5.1 (JAX-RS 2.0 RI) are also secured using OWSM policies. For more information about securing RESTful web services and clients built using Jersey 2.5.1 (JAX-RS 2.0 RI), see Securing RESTful Web Services and Clients in *Developing and Securing RESTful Web Services for Oracle WebLogic Server*.

The following sections describe how to develop RESTful web services:

- "Example of a Simple RESTful Client"
- "Creating RESTful Web Service Clients"

For information about:

- Developing RESTful web service clients, see Developing RESTful Web Services and Clients.
- Securing RESTful web service clients, see Securing RESTful Web Services.
- Administering RESTful web service clients, see *Administering Web Services*.

Example of a Simple RESTful Client

This example provides a simple RESTful client that calls the RESTful service defined in the Simple RESTful Service Client above. It uses classes that are provided by the Jersey 1.18 (JAX-RS 1.1 RI) specifically; they are not part of the JAX-RS standard.

> **Note:**
>
> Alternatively, you can create an instance of the `com.sun.jersey.api.client.Client` class. However, you will not be able to take advantage of the Oracle extensions, such as securing the RESTful client using Oracle Web Services Manager (OWSM) policies, as described in Attaching Policies to RESTful Web Service Clients.
>
> For the Jersey 2.5.1 (JAX-RS 2.0 RI), a standard client API is supported.

```
package samples.helloworld.client;

import weblogic.jaxrs.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/jersey/helloWorld");
      String response = resource.get(String.class);
   }
}
```

Creating RESTful Web Service Clients

You can create a new RESTful web service client from an existing remote or local WADL using the Create RESTful Proxy Client wizard.

To create a RESTful web service client:

1. In the Applications window, select the project containing the Java class from which you want to create a RESTful web service.
2. Choose **File > New > From Gallery** to open the New Gallery.
3. In the **Categories** list, expand **Business Tier** and select **Web Services**.
4. In the **Items** list, select **RESTful Client and Proxy** and click **OK** to launch the Create RESTful Proxy Client wizard.

   For detailed help about completing the wizard, press **F1** or click **Help** from within the wizard.

Creating WebSockets

WebLogic Server supports the WebSocket protocol (RFC 6455), which provides full-duplex communications between two peers over the TCP protocol.

The WebLogic Server implementation of the WebSocket protocol and its accompanying API enable you to develop and deploy applications that communicate bidirectionally with clients. To initiate the WebSocket connection, the client sends a handshake request to the server. The connection is established if the handshake request passes validation, and the server accepts the request. When a WebSocket connection is created, a browser client can send data to a WebLogic Server instance while simultaneously receiving data from that server instance.

Although you can use the WebSocket protocol for any type of client-server communication, the implementation is most commonly used to communicate with browsers running Web pages that use the World Wide Web Consortium (W3C) JavaScript WebSocket API. As part of the HTML5 specification (`http://www.w3.org/TR/html5/`), the WebSocket protocol is supported by most browsers. The WebLogic Server implementation of the WebSocket protocol also supports Java clients.

The JSR 356 Java API for WebSocket reference implementation RI, supported by WebLogic Server, enables you to create, configure, and deploy WebSocket endpoints in web applications. The API includes the following packages:

- `javax.websocket.server`—Create and configure server endpoints.
- `javax.websocket`—Create clients and configure client and server endpoints.

For more information about the WebSocket protocol and its support on WebLogic Server, see:

- "Using the WebSocket Protocol in WebLogic Server" in *Developing Applications for Oracle WebLogic Server*
- WebSocket protocol (RFC 6455): `http://tools.ietf.org/html/rfc6455`

The following sections provide more information about configuring WebSockets using JDeveloper:

- How to Configure WebSockets in the Properties Window
- How to Configure WebSockets Using Annotations
- How to Test the WebSocket Connection

How to Configure WebSockets in the Properties Window

You can configure WebSockets in the Properties window.

To configure WebSockets in the Properties window:

1. Create a WebSocket project.

   For more information about creating a project with preconfigured features, see How to Create a Project.

2. Create a Java class.

   For more information, see How to Create a New Java Class or Interface.

3. With the Java class open in the source editor, choose **Window > Properties** to open the Properties window.

   For more information at any time, press **F1** or click **Help** from within the Properties window.

4. With the cursor in the public class, navigate to the **Web Socket** node in the Properties window.
5. Select:
   - **Web Socket Client** to configure a WebSocket client. In this case, the service is updated to include the `@ClientEndpoint` annotation and import the `javax.websocket.ClientEndpoint` API.

     Web Socket Service to configure a WebSocket service. In this case, the service is updated to include the `@ServiceEndpoint` annotation and import the `javax.websocket.server.ServerEndpoint` API. By default, the URI template endpoint name is set to `service`.
6. Optionally, configure the WebSocket endpoint properties defined in the following table. When you configure a property, the attribute is added to the `@ClientEndpoint` or `@ServiceEndpoint` annotation for WebSocket clients and services, respectively.

Table 21-9 WebSocket Server Endpoint Properties

Property	Annotation Attribute	Description
URI Template	`value`	Relative URI path at which the server endpoint will be deployed. **Note**: This property is available for WebSocket services only.
Custom Configurator	`configurator`	Custom implementation of `ServerEndpointConfiguration.Configurator`.
Decoders	`decoders`	List of message decoder class names.
Encoders	`encoders`	List of message encoder class names.
Supported Sub Protocols	`subprotocols`	List of supported subprotocols.

How to Configure WebSockets Using Annotations

The Java API for WebSocket (JSR-356) enables you to create, configure, and deploy WebSocket endpoints in web applications. The WebSocket client API specified in JSR-356 also enables you to access remote WebSocket endpoints from any Java application.

The process for creating and deploying a WebSocket endpoint is as follows, as described in Using the WebSocket Protocol in WebLogic Server in *Developing Applications for Oracle WebLogic Server*:

1. Create an endpoint class.

   The Java API for WebSocket enables you to create the following kinds of endpoints:
   - Annotated endpoints
   - Programmatic endpoints

   For more information, see Creating an Endpoint.
2. Implement the lifecycle methods of the endpoint.

   Add a method to your endpoint class to handle an event and annotation that method to designate it as a handler. For more information, see Handling Life Cycle Events for a WebSocket Connection.
3. Add your business logic to the endpoint.
4. Deploy the endpoint inside a web application.

For more information, see Deploying a WebSocket Application.

5. Create a WebSocket client application.

   For more information, see Writing a WebSocket Client.

For more information about and examples of creating and configuring WebSocket service and client applications using the WebSocket API, see Using the WebSocket Protocol in WebLogic Server.

How to Test the WebSocket Connection

When you run the WebSocket application, you can test the connection by clicking the Target URL provided in the integrated application server log window. If the WebSocket connection is accessible, the following message is displayed:

```
Connected successfully
```

Attaching Policies

Learn about OWSM and Oracle WebLogic Web Service policies and how to use them with web services.

The following sections describe how to attach policies to web services and clients, and configure the policy repository to use custom web service policies.

- What You May Need to Know About OWSM Policies
- What You May Need to Know About Oracle WebLogic Web Service Policies
- How to Attach Policies to JAX-WS Web Service and Clients
- How to Attach Policies to RESTful Web Services and Clients
- How to Use a Different OWSM Policy Store
- How to Use Custom Web Service Policies

What You May Need to Know About OWSM Policies

OWSM policies can be attached to JAX-WS web services at the port level and RESTful web services at the servlet level.

> **Note:**
>
> You can attach OWSM policies to RESTful web services and clients that are built using Jersey 1.*x* JAX-RS RI. RESTful web services and clients that are built using Jersey 2.5.1 (JAX-RS 2.0 RI) are secured using OWSM policies. See Securing RESTful Web Services and Clients.

JDeveloper is preconfigured to use the OWSM policy store set at the default location in the WS Policy page of the Preferences dialog, which you can access in one of the following ways:

- **Tools > Preferences > WS Policy Store**
- **Application > Application Properties > WS Policy Store**

You can specify another policy store location to use your organization's custom OWSM policies. See How to Use a Different OWSM Policy Store and *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

> **Note:**
>
> A web service Provider must be deployed with a WSDL file in order to ensure any attached OWSM policies are enforced. You can the WSDL to the web service provider using the `wsdlLocation` element to the `@WebServiceProvider` annotation.

What You May Need to Know About Oracle WebLogic Web Service Policies

Oracle WebLogic web service policies can be attached to JAX-WS 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, for example, to be applied on the inbound (request) message or outbound (response) message, or both.

You can configure JDeveloper to use the custom Oracle WebLogic web service policies of your organization. See How to Use Custom Web Service Policies.

Jersey 2. *x* (JAX-RS 2.0 RI) web services are secured using Oracle Web Services Manager (OWSM) security policies. See *Securing WebLogic Web Services for Oracle WebLogic Server*.

How to Attach Policies to JAX-WS Web Service and Clients

This section describes how to attach policies to JAX-WS web services and clients created in JDeveloper. You can use the following types of policies:

- Oracle Web Service Manager (OWSM) policies—You can attach security policies only.
- Oracle WebLogic web service policies

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 OWSM policies and later decide that you want to use Oracle WebLogic web service policies, you must delete the OWSM policies before you can attach the Oracle WebLogic web service policies.

The following sections describe how to use policies with web services:

- Attaching Policies to JAX-WS Web Services
- Attaching OWSM Policies to JAX-WS Web Service Clients
- Overriding OWSM Policy Configuration Properties for the JAX-WS Web Service Clients
- Invoking JAX-WS Web Services Secured Using WebLogic Web Service Policies
- Editing or Removing Policies from JAX-WS Web Services

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 Predefined Policies to Use in the *Securing Web Services and Managing Policies with Oracle Web Services Manager*.
2. Attach the policies to a class or service. For more information, see Attaching Policies to JAX-WS 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 Securing Web Services.
4. Test the web service to ensure that the policies work as expected. For more information, see How to Test Web Services in a Browser.

Attaching Policies to JAX-WS Web Services

JDeveloper allows you to attach Oracle Web Service Manager (OWSM) 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 Securing Web Services in the *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

You can attach policies to web services as described in the following sections:

- Attaching Policies in the Web Service Wizard
- Attaching Policies Using Annotations
- Attaching Policies in the Properties Window

Attaching Policies in the Web Service Wizard

You can attach policies to web services by setting 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.

For more information about creating web services using the Create Java Web Service Wizard, see Creating Java Web Services .

To attach policies in the web service wizard:

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 OWSM 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.

Attaching Policies Using Annotations

You can attach policies to web services by adding policy annotations directly in the Java class. The Code Insight feature can help you, as described inHow to Work with Web Services Code Insight.

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, `@SecurityPolicy`. When you pause, or click **Alt+Enter,** a popup appears from which you can choose the correct entry to complete the statement.

Annotations for Attaching OWSM Policies

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 the `weblogic.wsee.jws.jaxws.owsm.SecurityPolicies` annotation 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, enter **uri=""**, place your cursor within the empty quotes, and click **Ctrl+Alt+Spacebar**.

For more information about using the policy annotations, see Attaching Policies to Java EE Web Services and Clients Using Annotations in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

Annotations for Attaching WebLogic Web Service Policies

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 the `weblogic.jws.Policies` annotation 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, enter the **uri=""** argument, place your cursor within the empty quotes, and click **Ctrl+Alt+Spacebar**.

For more information about the policy annotations, see Updating the JWS File with @Policy and @Policies Annotations in *Securing WebLogic Web Services for Oracle WebLogic Server*.

Attaching Policies in the Properties Window

You can attach policies to web services using the Properties window.

To attach policies in the Properties window:

1. With the web service class open in the source editor, choose **Window > Properties** to open the Properties window.

   For more information at any time, press **F1** or click **Help** from within the Properties window.

2. With the cursor in the `public class` or `@WebService` line of the class, navigate to the Policies node where you can choose to use OWSM Policies or Oracle WebLogic web service policies.

3. Select **Secure with OWSM Policies** or **Secure with WLS Policies**.

   The Properties window is refreshed to display options to select single or multiple policies for the policy type selected (OWSM or WLS).

   > **Note:**
   >
   > 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.

4. Select a single policy from the **Single Policy** list, or click **...** to attach multiple policies from the Edit Property: Multiple Policies dialog.

   When using the Edit Property: Multiple Policies dialog box to attach multiple OWSM policy files, click **View** to review the policy assertions in the policy file.

Attaching OWSM Policies to JAX-WS Web Service Clients

JDeveloper allows you to attach OWSM policies to JAX-WS web service clients.

After you attach an OWSM policy to a web service client, you need to configure the policies. For more information, see Securing Web Services in the *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

> **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 *Securing WebLogic Web Services for Oracle WebLogic Server*.

You can attach OWSM policies to web service clients by:

- Attaching OWSM Policies in the Web Service Client and Proxy Wizard
- Attaching OWSM Policies Using Annotations
- Attaching OWSM Policies Using Feature Classes

Attaching OWSM Policies in the Web Service Client and Proxy Wizard

You can attach policies to web service clients by setting the policies to attach in the web service client and proxy wizard when creating a new web service client or in the web service client editor when updating a web service client that already exists.

For more information about creating web services using the Create Web Service Client an Proxy wizard, see Creating the Client and Proxy Classes.

To attach OWSM policies in the web service client:

In the Create Web Service Client an Proxy wizard or client editor, navigate to the Client Policy Configuration page. For more information at any time, press **F1** or click **Help** from within the dialog.

When attaching OWSM 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.

Attaching OWSM Policies Using Annotations

You can attach policies to web service clients by adding policy annotations directly to the injection target (`@WebServiceRef`), using one of the following annotations:

- `@weblogic.wsee.jws.jaxws.owsm.SecurityPolicy` annotation to attach a single policy
- `@weblogic.wsee.jws.jaxws.owsm.SecurityPolicies` annotation to attach multiple policies

The following shows an example of attaching OWSM security policies to web service clients using annotations.

```
package wsrm_jaxws.example;
import java.xml.ws.WebService;
import java.xml.ws.WebServiceRef;
import weblogic.wsee.jws.jaxws.owsm.SecurityPolicy;
import weblogic.wsee.jws.jaxws.owsm.SecurityPolicies;
import oracle.wsm.security.util.SecurityConstants.ClientConstants;
...
@WebServiceRef(name="MyServiceRef")
@SecurityPolicies({
   @SecurityPolicy(uri="policy:oracle/wss10_message_protection_client_policy"),
   @SecurityPolicy(uri="policy:oracle/authorization_policy")
})
Service service;
...
```

For more information, see Attaching Policies to Java EE Web Services and Clients Using Annotations in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

> **Note:**
>
> Attaching OWSM policies using Feature classes, as described in Overriding OWSM Policy Configuration Properties Using RequestContext, takes precedence over annotations.

To attach OWSM policies using annotations:

1. Open the web service client in the source editor.
2. Position your cursor in the injection target line, right-click the mouse, and select **Add JEE Client Policy Annotations** from the context menu.

   The Client Policy Configuration window is displayed that contains a list of OWSM security policies.
3. Select the security policies that you wish to attach to the client.

   When attaching OWSM 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.
4. Click **OK**.

   The client code is updated with the appropriate annotations.

Attaching OWSM Policies Using Feature Classes

You can attach policies to web services by manually adding one of the following feature classes to the web service client:

- `weblogic.wsee.jws.jaxws.owsm.SecurityPolicyFeature` class to attach a single policy
- `weblogic.wsee.jws.jaxws.owsm.SecurityPoliciesFeature` to attach multiple policies

The following example shows how to use the `SecurityPolicyFeature` class to attach an OWSM policy to a web service client.

```
...
JAXWSService jaxWsService = new JAXWSService ();
weblogic.wsee.jws.jaxws.owsm.SecurityPolicyFeature securityFeature = new
weblogic.wsee.jws.jaxws.owsm.SecurityPolicyFeature {
new weblogic.wsee.jws.jaxws.owsm.SecurityPolicyFeature("policy:oracle/
wss_username_token__over_ssl_client_policy") };

JAXWSServicePort  port =  jaxWsService.getJaxWsServicePort(securityFeature);
...
```

The following example shows how to use the `SecurityPoliciesFeature` class to attach multiple OWSM policy to a web service client.

```
...
weblogic.wsee.jws.jaxws.owsm.SecurityPoliciesFeature
```

```
securityFeature = new weblogic.wsee.jws.jaxws.owsm.SecurityPoliciesFeature
(new String[] { "policy:oracle/wss_username_token_over_ssl_client_policy",
"policy:oracle/authorization_policy"});
```

For more information, see Attaching Policies to Java EE Web Service Clients Using Feature Classes in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

> **Note:**
>
> Attaching OWSM policies using Feature classes takes precedence over annotations (described in Overriding OWSM Policy Configuration Properties Using Annotations).

Overriding OWSM Policy Configuration Properties for the JAX-WS Web Service Clients

JDeveloper allows you to override configuration properties for policies attached to JAX-WS web service clients.

You can attach OWSM policies to web service clients by:

- Overriding OWSM Policy Configuration Properties in the Web Service Client and Proxy Wizard
- Overriding OWSM Policy Configuration Properties Using Annotations
- Overriding OWSM Policy Configuration Properties Using RequestContext

Overriding OWSM Policy Configuration Properties in the Web Service Client and Proxy Wizard

When attaching policies to web service clients using the web service client and proxy wizard, on the Client Policy Configuration page, select **Override Properties...** to display the Set Override Properties dialog. The configuration properties for the selected policies are displayed. Set the values as desired, and click OK.

In the Create Web Service Client an Proxy wizard or client editor, navigate to the Client Policy Configuration page, select **Override Properties...**. The Set Override Properties dialog displays enabling you to edit the configuration properties for the selected policies. For more information at any time, press **F1** or click **Help** from within the dialog.

For more information about creating web services using the Create Web Service Client an Proxy wizard, see Creating the Client and Proxy Classes.

Overriding OWSM Policy Configuration Properties Using Annotations

You can override the default configuration properties of an OWSM security policy programmatically at design time using the `@Property` annotation when attaching an OWSM security policy using the `@SecurityPolicy` annotation. For more information, see Attaching Policies to Java EE Web Services and Clients Using Annotations in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

The following example shows how to attach a policy to a web service client using annotations. In this example, the `@Property` annotation is used to override the keystore recipient alias configuration property when attaching the client policy.

```
package wsrm_jaxws.example;
import java.xml.ws.WebService;
```

```
import java.xml.ws.WebServiceRef;
import weblogic.wsee.jws.jaxws.owsm.SecurityPolicy;
import weblogic.wsee.jws.jaxws.owsm.SecurityPolicies;
import oracle.wsm.security.util.SecurityConstants.ClientConstants;
...
@WebServiceRef(name="MyServiceRef")
@SecurityPolicies({
   @SecurityPolicy(uri="policy:oracle/wss10_message_protection_client_policy",
          properties = {
             @Property(name="ClientConstants.WSS_KEYSTORE_LOCATION",
                value="c:/mykeystore.jks")
          }
   ),
   @SecurityPolicy(uri="policy:oracle/authorization_policy")
})
Service service;
...
```

Overriding OWSM Policy Configuration Properties Using RequestContext

You can attach policies to web services by manually adding one of the following feature classes to the web service client:

- `weblogic.wsee.jws.jaxws.owsm.SecurityPolicyFeature` class to attach a single policy
- `weblogic.wsee.jws.jaxws.owsm.SecurityPoliciesFeature` to attach multiple policies

For more information, see Overriding Client Policy Configuration Properties at Design Time in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

Invoking JAX-WS 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 invoke web services secured using WebLogic web service policies:

1. Perform one of the following tasks:
   - Create a web service client. For more information, see Creating the Client and Proxy Classes.
   - Edit a web service client. For more information, see Editing 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 = "<username>";
   String password = "<password>";
   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 *Securing WebLogic Web Services for Oracle WebLogic Server*.

Editing or Removing Policies from JAX-WS Web Services

You can edit policies and remove them entirely from web services with either of the following:

- Web service editor, as described in Editing or Removing Policies Using the Web Service Editor.
- Source editor, as described in Editing or Removing Policies Using Annotations in the Java Class.
- Properties window, as described in Editing or Removing Policies Using the Properties Window.

Editing or Removing Policies Using the Web Service Editor

You can edit or remove policies using the web service editor.

To edit or remove policies using the web service editor:

1. Right-click the web service in the Applications window, 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 OWSM 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.

Editing or Removing Policies Using Annotations in the Java Class

You can edit or remove policies using annotations in the Java class in the source editor.

To edit or remove policies using annotations in the Java class, pen the web service class in the source editor, where the Code Insight feature is available to help you. For more information, see How to Work with Web Services Code Insight. Edit or remove the annotations, as required.

Editing or Removing Policies Using the Properties Window

You can edit or remove policies using the Properties window.

To edit or remove policies using the Properties window:

1. With the JAX-WS web service class open in the source editor, choose **Window > Properties** to open the Properties window.

   For more information at any time, press **F1** or click **Help** from within the Properties window.
2. With the cursor in the `public class` or `@WebService` line of the class, navigate to the Policies node:
   - To edit multiple policies, click **...** to open the Edit Property: Multiple Policies dialog.
   - To edit a single policy, delete the name 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.

How to Attach Policies to RESTful Web Services and Clients

This section describes how to attach policies to RESTful web services and clients created in JDeveloper.

- Attaching Policies to RESTful Web Services
- Attaching Policies to RESTful Web Service Clients
- Editing or Removing Policies from RESTful Web Services and Clients

Attaching Policies to RESTful Web Services

To secure RESTful web services, you can attach one of the OWSM predefined security policies described in Which OWSM Policies Are Supported for RESTful Web Services?.

For information about:

- Securing RESTful web services, see Securing RESTful Web Services and Clients Using OWSM Policies in *Developing and Securing RESTful Web Services for Oracle WebLogic Server*.
- Supported security policies, see Predefined Policies.
- Configuring the security policies, see Securing Web Services.

  Both the above two links are in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

To attach policies to RESTful web services:

1. In the Applications window, right-click the **web.xml** file located in the Web Content > WEB-INF folder.
2. Select **Secure RESTful Application**.

   The Select Policy From List dialog opens.
3. Select the security policy from the list.

   You can attach only one authentication policy and one authorization policy.
4. Click **OK**.

The security policy configuration is saved to the `wsm-assembly.xml` deployment descriptor file. If the `wsm-assembly.xml` file does not exist, it will be created.

Attaching Policies to RESTful Web Service Clients

To secure RESTful web services, you can attach one of the OWSM predefined security policies described in Which OWSM Policies Are Supported for RESTful Web Services? in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

You can attach OWSM policies to the RESTful client using one of the methods described in the following sections:

- Attaching OWSM Policies in the RESTful Client and Proxy Wizard
- Attaching OWSM Policies in the Client Policy Configuration Dialog
- Attaching OWSM Policies to RESTful Clients Programmatically

For more information about:

- Securing RESTful web services, see Securing RESTful Web Services and Clients Using OWSM Policies in *Developing and Securing RESTful Web Services for Oracle WebLogic Server*.
- Supported security policies, see Predefined Policies.
- Configuring the security policies, see Securing Web Services.

Attaching OWSM Policies in the RESTful Client and Proxy Wizard

You can attach policies to web service clients by setting the policies to attach in the RESTful client and proxy wizard when creating a new RESTful client.

For more information about creating web services using the Create RESTful Proxy Client wizard, see Creating RESTful Web Service Clients.

To attach OWSM policies to the RESTful client in the RESTful Client and Proxy Wizard:

In the Create RESTful Proxy Client wizard, navigate to the Client Policy Configuration page. For more information at any time, press **F1** or click **Help** from within the dialog.

When attaching OWSM 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.

Attaching OWSM Policies in the Client Policy Configuration Dialog

You can attach policies to web service clients by setting the policies to attach in the Client Policy Configuration dialog.

To attach OWSM policies to the RESTful client in the Client Policy Configuration dialog:

1. In the Applications window, right-click the RESTful web service client file located in the Application Sources folder.

2. Select **Secure RESTful Client**.

   The Client Policy Configuration dialog opens.

3. Select the security policy from the list.

   You can attach only one authentication policy and one authorization policy.

4. Click **OK**.

   The `customizeClientConfiguration` method is updated in the client to include the policy attachment.

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

When attaching OWSM 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.

Attaching OWSM Policies to RESTful Clients Programmatically

OWSM security policies can only be attached programmatically to RESTful web service clients, as described in “Attaching Policies to RESTful Web Service Clients Using Feature Classes" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

Using the `com.sun.jersey.api.client.config.ClientConfig`, you can attach OWSM policies and override configuration properties, as shown in the following example. The following code attaches the `oracle/wss_http_token_client_policy` policy to the client, and overrides the `CO_CSF_KEY` configuration property with the value `weblogic-csf-key`.

```
package samples.helloworld.client;

import weblogic.jaxrs.api.client.Client;
import com.sun.jersey.api.client.WebResource;
import com.sun.jersey.api.client.config.ClientConfig;
import com.sun.jersey.api.client.config.DefaultClientConfig;
import oracle.wsm.metadata.feature.PolicyReferenceFeature;
import oracle.wsm.metadata.feature.AbstractPolicyFeature;
import oracle.wsm.metadata.feature.PolicySetFeature;
import oracle.wsm.metadata.feature.PropertyFeature;
...
    public static void main(String[] args) {
        ClientConfig cc = new DefaultClientConfig();
        cc.getProperties().put(AbstractPolicyFeature.ABSTRACT_POLICY_FEATURE,
             new PolicySetFeature(
                new PolicyReferenceFeature((
                  "oracle/wss_http_token_client_policy"), new
                  PropertyFeature(SecurityConstants.ConfigOverride.CO_CSF_KEY,
                      "weblogic-csf-key"))));
        Client c = Client.create(cc);
...
```

Overriding OWSM Policy Configuration Properties for the RESTful Web Service Clients

JDeveloper allows you to override configuration properties for policies attached to RESTful web service clients.

You can override OWSM policy configuration properties for RESTful web service clients by:

- Overriding OWSM Policy Configuration Properties in the Web Service Client and Proxy Wizard
- Overriding OWSM Policy Configuration Properties Using Annotations

Overriding OWSM Policy Configuration Properties in the Web Service Client and Proxy Wizard

When attaching policies to RESTful web service clients using the RESTful client and proxy wizard or client editor, navigate to the Client Policy Configuration page, select one or more policies to attach, and click **Override Properties...**. The Set Override Properties dialog displays enabling you to edit the configuration properties for the selected policies. For more information at any time, press **F1** or click **Help** from within the dialog. Set the values as desired, and click **OK**.

For more information about creating RESTful web service clients using the Create RESTful Proxy Client wizard, see Creating RESTful Web Service Clients.

Overriding OWSM Policy Configuration Properties for RESTful Clients Using Feature Classes

When creating a RESTful web service client, when attaching an OWSM security policy programmatically, you can override the default configuration properties of an OWSM security policy using the `oracle.wsm.metadata.feature.PropertyFeature` class. For more information, see “Attaching Policies to RESTful Web Service Clients Using Feature Classes" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

Editing or Removing Policies from RESTful Web Services and Clients

You can edit policies and remove them entirely from your RESTful web services and clients, as described in the following sections:

- Editing or Removing Policies from RESTful Web Services
- Editing or Removing Policies from RESTful Web Service Clients

Editing or Removing Policies from RESTful Web Services

To edit or remove policies from RESTful web services:

1. In the Applications window, right-click the **web.xml** file located in the Web Content > WEB-INF folder.
2. Select **Secure RESTful Application**.

   The Select Policy From List dialog opens.
3. Select the security policy from the list.

   You can attach only one authentication policy and one authorization policy.
4. Click **OK**.

The security policy configuration is saved to the `wsm-assembly.xml` deployment descriptor file. If the `wsm-assembly.xml` file does not exist, it will be created.

Editing or Removing Policies from RESTful Web Service Clients

Use any of the methods described in Attaching Policies to RESTful Web Service Clients to edit or remove policies from the RESTful web service client.

How to Use a Different OWSM Policy Store

The OWSM policy store is installed as part of JDeveloper. You can configure 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 use a different OWSM policy store:

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 **Connections** 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 OWSM Policy Manager. To verify that the OWSM 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 OWSM Policy Manager is operational, then a list of the predefined policies is displayed with descriptions. For more information about troubleshooting the OWSM Policy Manager, see “Diagnosing Problems with Oracle OWSM" in the *Administering Web Services*.

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 OWSM policies or Oracle WebLogic web service policies, as described in the following sections:

- Using Custom OWSM Policies
- Using Custom Oracle WebLogic Web Service Policies

Using Custom OWSM Policies

To use custom OWSM policies:

- Add a custom policy in the default policy store location at:

`JDEV_USER_HOME\system14.1.2.0.x.x.x\DefaultDomain\store\gmds\owsm`

If not set, `JDEV_USER_HOME` defaults to `C:\Users\user-dir\AppData\Roaming\JDeveloper`.

Within this directory, policies must be included using one of the following directory structures:

– Predefined OWSM policies: `policies/oracle/`*`policyname`*

– Custom user policies: `policies/`*`policyname`*

> **Note:**
>
> When exporting policy files from the OWSM 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 extract the policies from the zip archive into the JDeveloper policy store using the required directory structure noted above. Otherwise, the policies will not be available in the JDeveloper environment. For information about exporting policies from the OWSM repository, see Understanding the Different Mechanisms for Importing and Exporting Policies.

- Specify a different policy store.
  If you elect to use a policy store on a remote application server, you can import custom OWSM policies to the MDS repository on the remote application server using Fusion Middleware Control or WLST. See Understanding the Different Mechanisms for Importing and Exporting Policies.

For information about creating custom policies, see Creating Custom Assertions.

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.

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, as described in How to Deploy Web Services to Integrated WebLogic Server.
- Deploy web services to Oracle WebLogic Server or other application server, as described in How to Deploy Web Services to a Standalone Application Server.
- Deploy web services to an archive file, as described in Deploying to a Java JAR.
- Define a different WebLogic Server domain to be the Integrated WebLogic Server on which to run web services and deploy web services to it, as described in Running Java EE Applications in the Integrated Application Server.
- Undeploy a web service, as described in How to Undeploy Web Services.

How to Deploy Web Services to Integrated WebLogic Server

You can deploy a web service generated in JDeveloper to Integrated WebLogic Server. For more information, see Running Java EE Applications in the Integrated Application Server.

To deploy a web service to Integrated WebLogic Server:

1. If not already started, start the Integrated WebLogic Server.

> **Note:**
>
> The first time you start the Integrated WebLogic Server, a dialog is displayed prompting you to enter a password for the administrator ID for the default domain. When you click **OK**, the default domain is created and the Integrated WebLogic Server is started. You only need to do this once.

2. In the Applications window, right-click the project containing the web service, and choose **Deploy > Web Services**.
3. In the Deploy Web Services dialog, on the Deployment Action page, select **Deploy to Application Server** and click **Next**.
4. On the Select Server page, select **IntegratedWebLogicServer** and click **Next**.
5. On the WebLogic Options page, configure the WebLogic Server deployment options, specifying:
   - Whether to deploy the application to all or select instances in the domain. By default the application is deployed to all instances.
   - Whether the application is registered as a shared library. By default, it is registered as a standalone application.
6. Click **Next** to view the Summary page or **Finish** to deploy the web services.

How to Deploy Web Services to a Standalone Application Server

You can deploy a web service generated in JDeveloper to one of the following standalone application servers:

- Oracle WebLogic Server
- JBoss 5.x
- Tomcat 6.x
- WebSphere Server 7.x

To deploy a web service to a standalone application:

1. Ensure that the standalone application server is running.
2. In the Applications window, 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.

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 Resources window.

To undeploy a web service:

1. In the Application Server window, select the application server connection you have been using and expand **Web Services**.
2. Right-click `application-name_project-name_ws` and choose **Undeploy**.

Testing and Debugging Web Services

Learn about the tools in JDeveloper that help you to test and debug web services.

Developer provides a number of ways that you can test and debug web services:

- Run a JAX-WS web service deployed to Integrated WebLogic Server in a browser to check that it returns what you expect, as described in How to Test Web Services in a Browser.
- 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, as described in How to Debug Web Services.
- Install and use JUnit, as described in How to Test Web Services with JUnit.
- View web service message logs, as described in How to View Web Service Message Logs for an Application Server.

In addition to the methods described in this section, you can use the HTTP Analyzer to examine the content of web services over HTTP, similar to examining other packet information. For more information, see Monitoring Web Services Network Traffic .

How to Test Web Services in a Browser

Once you have created and deployed a web service, you can test it to ensure that it returns what you expect by running it in the browser using the Web Services Test Client. For more information about using the Web Services Test Client to test web services, see “Testing Web Services" in *Administering Web Services*.

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.

JDeveloper lets you debug a web service that is running in the Integrated WebLogic Server locally, or a web service that is deployed remotely.

As you debug your web service, JDeveloper also provides a number of special-purpose debugging windows to help you analyze your code and identify problem areas. You can open the debugger windows by choosing **Window** > **Debugger**, then choosing a window from the menu. The Data window is particularly useful for debugging web services. For more information, see Using the Data Window for Debugging Web Services.

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 How to Examine Web Services using the HTTP Analyzer .

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 using one of the following methods:

- Insert breakpoints in the web service class, then run a proxy client against it. This allows you to debug the service class itself.
- Insert 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, as described in Disabling the Use of a Proxy Server When Accessing the Internet.
2. Run the web service in debug mode. In the Applications window, right-click the web service, and choose **Debug**.

   The Integrated WebLogic Server is started (or restarted) in debug mode, and the web service is deployed to it. The results are displayed in the log window.
3. Create a web service client, as described in How to Create JAX-WS Web Service Clients.

   A proxy container is generated and displayed in the Applications window, with a Java class called *webservice*`PortClient.java` displayed in the source editor.
4. In the source editor, locate the comment `// Add your own code to call the desired methods`, and enter some code.
5. 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 Running and Debugging Java Projects .

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 and 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 Deploying Web Services.
3. Create a web service client, as described in How to Create JAX-WS Web Service Clients.
4. In the source editor, locate the comment `// Add your own code to call the desired methods`, 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 Running and Debugging Java Projects .

Using the Data Window for Debugging Web Services

JDeveloper provides a number of special-purpose debugging windows to help you analyze your code and identify problem areas. The Data window is particularly useful for debugging web services.

The Data window displays information about variables in your program for the current context. By default, the Data window displays local variable information while debugging a program. For web services, the Data window also provides additional information derived from the web service. For example, the returned values for the following method calls are displayed when debugging a proxy class (this is not an exhaustive list):

- `com.sun.xml.ws.client.RequestContext.getEndpointAddressString()`
- `javax.xml.ws.Binding.getBindingID()`
- `javax.xml.ws.BindingProvider.getResponseContext()`

To open the Data debugger window choose **Window** > **Debugger** > **Data**. For information about accessing and using the other debugging windows, see Using the Debugger Windows.

While debugging, you can use filters to reduce the number of fields that are displayed when you expand an object in the Data 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. See How to Show and Hide Fields in the Filtered Classes List.

How to Test Web Services with JUnit

JDeveloper provides support for JUnit regression testing for your web services. JUnit is an open source Java regression testing framework that is available as an optional feature in

JDeveloper. You can use JUnit to write and run tests that verify your code. For detailed information about JUnit, visit the JUnit web site, `http://www.junit.org`.

To use JUnit, you need to install the JUnit extension. After you install the JUnit extension, you can create a JUnit test class for the web services that you want to test.

To run a JUnit test on a web service:

1. Install the JUnit extension from the JDeveloper Help menu. For more information, see How to Install Extensions with Check for Updates.
2. Right-click a web service in the Applications window and choose **Create JUnit class for Web Service**. This will create a JUnit test class.
3. Edit the generated class.

   The generated test class can run in the JUnit runtime; however, in order to gather meaningful results, you need to instrument the test class (for example, with assertions) to reflect your environment. There is one generated test method in the generated JUnit class per web service method, which can each be run in the JUnit test harness.

By default, when the generated test is run, JUnit sets up the web service endpoint in a web service endpoint publisher. You can, however, remove the web service endpoint publisher from the generated JUnit test class and set the web service endpoint to any application server where the web service is pre-deployed.

How to View Web Service Message Logs for an Application Server

If you have attached a log policy to a web service that is deployed on an application server, the log policy writes the SOAP message to a message log. The web service message log can be accessed using the Application Servers window.

The application server log files are available under the Log Files folder under each application server node for both integrated and remote servers. Under each Log Files folder there is a OWSM Logs folder that contains the web service log files.

To view the message log of a web service deployed to an application server

1. If necessary, open the Application Servers window by choosing **Window > Application Servers**.
2. Expand the application server that the web service is deployed on.
3. Expand the **Log Files** folder to access the OWSM Logs folder.
4. Expand the **OWSM Logs** folder, then right-click a log file (for example, `diagnostic.log`) and choose **View** from the context menu.

   The log contents are displayed in read-only mode in the editor.

Monitoring Web Services Network Traffic

You can use the HTTP Analyzer to examine the network traffic of a client connecting to a JAX-WS or RESTful web service. For more information, see Using the HTTP Analyzer with Web Services.

The HTTP Analyzer enables 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.

- Test web services that are secured using policies; encrypted messages will be decrypted.

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 update the proxy settings. For more information, see How to Use Proxy Settings and .

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 Deploying Web Services.
3. Start the HTTP Analyzer by selecting **Tools > 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 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**.
6. You can examine the contents of the HTTP headers of the request and response packets to see the SOAP structure (for JAX-WS web services), the HTTP content, the WADL structure (for RESTful services), the Hex content or the raw message contents by choosing the appropriate tab at the bottom of the HTTP Analyzer Test window.
7. You can test web services that are secured using policies by performing one of the following tasks:
   - Select an existing credential from the **Credentials** list.

     JDeveloper delivers with a set of preconfigured credentials, `HTTPS Credential`.
   - Click **New** to create a new credential. In the Credential dialog, define the credentials to use in the HTTP Analyzer Test window. You can define one of the following credentials: HTTPS, username token, X509, or STS. For more information, see Using SSL with the HTTP Analyzer.

22

Deploying Applications

This chapter describes how to deploy JDeveloper applications to the JDeveloper integrated application server, or to a target application server, for example to Oracle WebLogic Server or to a third-party server.
This chapter includes the following sections:

- About Deploying Applications
- Running Java EE Applications in the Integrated Application Server
- Connecting and Deploying Java EE Applications to Application Servers
- Deploying Java Applications
- Deploying Java EE Applications
- Post-Deployment Configuration
- Testing the Application and Verifying Deployment
- Deploying from the Command Line
- Deploying Using Java Web Start
- Deploying Using Weblogic SCA Spring
- Troubleshooting Deployment

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 use JDeveloper to run an application in the integrated application server.

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 *Developing Fusion Web Applications with Oracle Application Development Framework*.

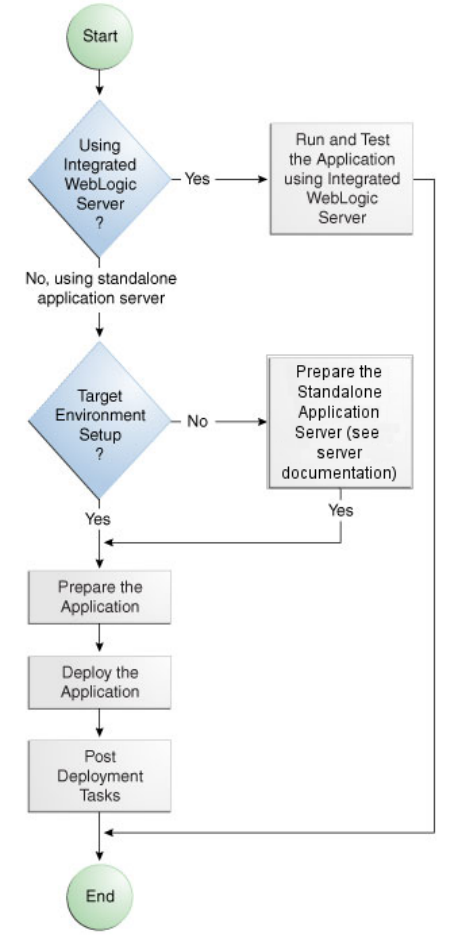
You can deploy applications in the following ways:

- 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, refer to the appropriate developer's guide for deployment information specific to the product.

The figure 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 appropriate documentation for a third-party application server.

Figure 22-1 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.

- Oracle Glassfish Server

  You can deploy to Oracle GlassFish Server

- A third-party application server, that is an application server provided by a vendor other than Oracle:
  - Apache Tomcat
  - IBM WebSphere
  - JBoss
  - GlassFish Server Open Source Edition

  For information about which server versions are compatible with JDeveloper, 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 integrated 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.

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 Applications window, or running an application by clicking **Run** on the JDeveloper main menu.

If you want to debug the application use the features described in Running and Debugging Java Projects .

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. See How to Create a Connection to the Target Application Server.
- Creating deployment profiles (if necessary). See How to Create and Edit Deployment Profiles.
- Creating deployment descriptors (if necessary, and that are specific to the target application server). How to Create and Edit Deployment Dependencies.
- Updating `application.xml` and `web.xml` to be compatible with the application server (if required). Viewing or Modifying Deployment Descriptor Properties.
- Migrating application-level security policy data to a domain-level security policy store. See Setting Up JDBC Data Sources on .

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.

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.
- Use Enterprise Manager, scripts, or the application server's administration tools to deploy the EAR file created in JDeveloper. See Deploying Oracle ADF Applications.

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.

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.

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.

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. See Understanding WebLogic Server Deployment Plans.

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. See Packaging the Extension.

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. JDeveloper then creates an in-memory deployment profile for deploying the application to the integrated application server. JDeveloper copies project and application 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 use the Application Server Properties dialog to edit the port you want the application to use. However you cannot specify a port number lower than 1024. This setting can be found on the Configuration page of the Application Server Properties dialog. Open the Application

Server Navigator, right-click **IntegratedWebLogicServer** and choose **Properties**, then click the **Configuration** tab and enter the desired port number (1024 or greater) in the **Preferred Port** field.

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 Servers window, right-click **IntegratedWebLogicServer** and choose **Create Default Domain**.

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`. See weblogic.xml Deployment Descriptor Elements.

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.

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 Applications window, 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.

How to Create a New Integrated Application Server Connection

To define an integrated server connection:

1. In the Application Servers window, 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.

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 Applications window, 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 Applications window, 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 Applications window. 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`.

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 Servers window.

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 Servers window by choosing **Window > Application Servers**.
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. If necessary, open the Application Servers window by choosing **Window > Application Servers**.
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, select Processes from the Window 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.

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 Running and Debugging Java Projects .

To run an application in the integrated application server:

1. In the Applications window, 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 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 Setting Up JDBC Data Sources on .

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 Applications window, 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 Applications window.

After the server has started, open the Processes window to display the integrated application server process. You can open the Processes window by choosing **Window > Processes** from the main menu.

To start an integrated application server:

1. If necessary, open the Application Servers window by choosing **Window > Application Servers**.
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 Servers window by choosing **Window > Application Servers**.
2. Right-click the Integrated WebLogic Server connection and choose **Debug Server Instance**.

Alternatively, choose **Run > Debug Server Instance** from the main menu.

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, In the Log Window, click the **Terminate** button and choose the profile or application you want to cancel.

How to Terminate an Integrated Application Server

After an integrated application server has started, the integrated application server process appears in the Processes window. For more information, see Understanding the Processes Window.

You can open the Processes window by choosing **Window > Processes** 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 **Window > Processes** 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 Servers window, 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.

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 Servers window by choosing **Window > Application Servers**.
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.

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 Servers window.
> - By running an application.

To launch and log in to the integrated application server Administration Console:

1. If necessary, open the Application Servers window by choosing **Window > Application Servers**.
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.

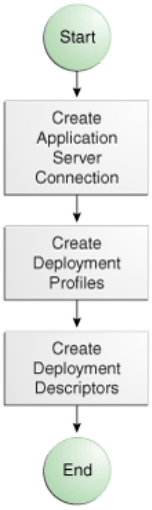
Connecting and Deploying Java EE Applications to Application Servers

Learn how to create connections to application servers, deployment profiles, and deployment descriptors.

Before you deploy an application to a standalone application server, you must perform prerequisite tasks within JDeveloper to prepare the application for deployment.

The figure shows 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 22-2 Preparing the Application for Deployment Flow Diagram



How to Create a Connection to the Target Application Server

You can deploy applications to the application server via JDeveloper application server connections.

This section describes how to generically create a connection to an application server. For information about connecting to a specific type of application server, see Connecting to Specific Application Server Types.

Before you begin:

- Ensure that the application server is installed and started.
- If you are working behind a proxy server you may want to be check the proxy settings that JDeveloper is using.
  1. Choose **Tools** > **Preferences** to open the Preferences dialog and navigate to the Web Browser and Proxy page. For more information at any time, press F1 or click **Help** in the Web Browser and Proxy page of the Preferences dialog.
  2. On the Proxy Settings tab you can see the proxy settings that JDeveloper is using, and if necessary change them. By default, JDeveloper uses the system default proxy settings. You can override these by choosing **No Proxy**, **Use Automatic Configuration Script**, or **Manual Proxy Settings** and entering your settings as appropriate. See Configuring Proxy Settings.

To create a connection to an application server:

1. Launch the Application Server Connection wizard in one of the following ways:
   - In the Application Servers window, right-click **Application Servers** and choose **New Application Server**.
   - In the New Gallery, expand **General**, select **Connections** and then **Application Server Connection**, and click **OK**.

- In the Resources window, choose **New** > **New Connection** > **Application Server**.

2. If the Usage page of the wizard is displayed, ensure that **Standalone Server** is selected and click **Next**.
3. In the Name and Type page, enter a name for your new connection.
4. In the Connection Type dropdown list, choose the type of application server you want to create a connection to. The options are:
   - **WebLogic 10.3**, **WebLogic 12.x**, or **WebLogic 14.x** to create a connection to Oracle WebLogic Server
   - **GlassFish 3.1** to create a connection to Oracle GlassFish Server or GlassFish Open Source Edition
   - **JBoss 5.x** to create a connection to JBoss
   - **Tomcat 7.x** to create a connection to Tomcat
   - **WebSphere Server 8.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 to access the application server.
7. Click **Next**.
8. On the Configuration page, enter the parameters to create a connection to the application server:
9. Click **Next**.
10. If you have chosen WebSphere, the JMX page appears where you enter JMX configuration values needed for deploying SOA applications.
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 Servers window.

> **Note:**
>
> To log in to the console, the server must be started.

1. In the Application Servers window, 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 user name 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 the 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.

Connecting to Specific Application Server Types

This section contains the specific information for accounting to different types of 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.

Connecting to Oracle WebLogic Server

To connect to Oracle WebLogic Server:

- The Oracle WebLogic host name is the name of the WebLogic Server instance containing the TCP/IP DNS where your application (`.jar`, `.war`, `.ear`, `.gar`) will be deployed.
- In the **Port** field, enter a port number for the Oracle WebLogic Server instance on which your application (`.jar`, `.war`, `.ear`, `.gar`) will be deployed.
  If you do not 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`, `.gar`) 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 do not 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.
- Optionally select **Always use Rest** to let JDeveloper leverage the WebLogic management REST APIs. This is required if you have network restrictions on using t3 traffic.

Connecting to GlassFish

If you are creating a secure connection to a GlassFish server, you must use the host name, for example `localhost`, instead of an IP address in **Host name** on the Configuration page, otherwise, the connection will fail. This is because the host name in the URL used to create the connection the server must match the host name in the certificate.

Connecting to JBoss

For JBoss:

- Enter or browse to the location of the JBoss deploy directory, where your application files (`.jar`, `.war`, `.ear`, `.gar`) 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.

Connecting to Tomcat

For Tomcat:

- In the **Webapps Directory** field enter or browse to the location of the webapps directory where you place the application `.war` files.

Connecting to WebSphere Server

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`, [.gar]) 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.

> **Note:**
>
> When the script file is specified you can encounter problems if the path contains a space. For example, `C:\Program Files (x86)\IBM\WebSphere\AppServer\profiles\AppSrv01\bin\wsadmin.bat` To work around this problem, use an OS short name for the directory with a space in the name (for example, Progra~1), or create a WebSphere application server instance in a path that does not have spaces and refer to the batch file there.

- 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.

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

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. All libraries marked Deploy by Default at the application level are deployed as an application library (in the `lib` of the EAR).
6. If an EJB Project A depends on the build output of Project B, the build output (for example, 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:

- **Business Components Archive:** Creates a simple archive file for deploying ADF Business Components.
- **Business Components Service Interface:** Creates a profile for deploying ADF Business Components as a service interface.
- **Client JAR File:** Used for deploying the standard Java EE client JAR file.
- **EJB JAR File:** Used to deploy the Java EE EJB module (EJB JAR). The EJB JAR contains the EJB components and the corresponding deployment descriptors.
- **Extension JAR:** Creates a profile for deploying an extension as a JAR file.
- **JAR file:** Creates a simple JAR archive from a project.
- **GAR file:** Creates an Oracle Coherence grid archive file. The GAR contains the artifacts of a Coherence application and includes a deployment descriptor.
- **MOF Model Library:** Creates MOF (Meta-Object Facility) Model Library JAR files which enable UML objects from one project to be reused by another.
- **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.

Creating Deployment Profiles

To create a deployment profile, select an application or a project then choose **File > New > From Gallery**, and from the General category, choose **Deployment Profile**. To create application level profiles, invoke the New Gallery at application level.

Other methods can be used to create a deployment profile:

- Use the Application Properties Deployment dialog:
  - From the JDeveloper toolbar, select **Application > Deploy > New Deployment Profile...**
  - Right-click on an application and select Deploy > New Deployment Profile... from the context menu.
  - In the Applications window, open the dropdown list on the Applications window toolbar. and choose **Deploy**.
- Use the Project Properties Deployment dialog:
  - Select the project in the Applications window, then from the menu, choose **Application > Project Properties**. Choose Deployment, and in the Deployment panel on the right, click the New Profile icon on the menu bar.
  - The context menu of a project in the Applications window.

To modify an existing deployment profile:

- Right-click the project in the Applications window 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 Applications window and choose **Application Properties** then choose **Deployment** in the tree structure in the wizard, then select the deployment profile and choose **Edit**.

To activate profile deployment:

- For a project level deployment profile, right-click the project in the Applications window then choose **Deploy > deployment profile.**
- For an application deployment profile, right-click the application in the Applications window then choose **Deploy > deployment profile**. Alternatively,
  - Right-click the application in the Applications window 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 Applications window 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 Applications window, right-click the application and choose **New**.

   For a project level deployment profile, in the Applications window, 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.
   - (Optional) Select **Platform**, select the application server you are deploying to, and then select the target application connection from the **Target Connection** dropdown list.
     By default, deployment will set this platform at deployment time to match the server you are deploying to. When you are deploying to a file, the default will be the "default platform" which is the latest Oracle Weblogic server release, in this case, Oracle WebLogic Server 14.x.

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.

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.

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 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.

How to Create and Edit Deployment Dependencies

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 application 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.

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.

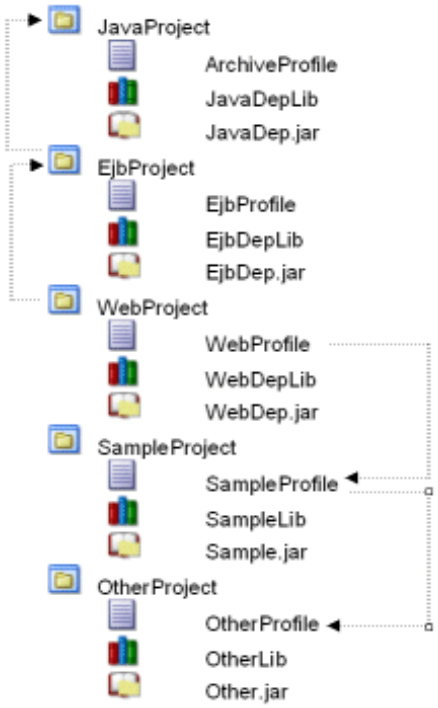
In an application, dependent libraries can be in the following projects:

- 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 22-3 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)

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.

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.

How to Create and Edit 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 22-1 provides a description of these files and illustrates how they relate to one another.

Table 22-1 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`. See Client Application Deployment Descriptor Elements.
`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`. See The weblogic-application.xml File.

`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`.
	`persistence-configuration.xml`
	For EJB 3.*x* modules. The format of this file is defined in `persistence-configuration.xsd`.
	For more information, see EJB Deployment Descriptors.
	`weblogic-cmp-rdbms-jar.xml`

`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`. See weblogic-ra.xml Schema.
`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`. See weblogic.xml Deployment Descriptor Elements.

Table 22-1 (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`.
	See Configuring WebLogic JDBC Resources.
	`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.
	See Understanding WebLogic Server Deployment Plans.
	`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.
	See Configuring Instrumentation.
	`weblogic-jms.xml`
	The format of this file is defined by `weblogic-jms.xsd`.
	Used to configure JMS drivers in the Oracle WebLogic Server.
	See Understanding WebLogic JMS.
	`weblogic-webservices.xml`
	The format of this file is defined by `weblogic-webservices.xsd`.
	See WebLogic Web Service Deployment Descriptor Schema Reference.

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 Applications window, 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 cannot 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. 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 Applications window. This ensures that the application is created with the correct descriptors.

To inspect or change deployment descriptor properties:

1. In the Applications window, 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.

How to Create a Web Service Policy Reference

Follow these steps to create a web service policy reference:

1. Select **File** > **New** > **WebLogic Deployment Descriptor**. From the General category, select Deployment Descriptors; choose the Weblogic item in the right-hand pane and click **OK**. The wizard opens.
2. On the Select Descriptor page, scroll down to select `webservice-policy-ref.xml`.
3. On the Select Version page, select the Deployment Descriptor version and click **Next**.
4. On the Summary page, click **Finish**.
5. The new configuration file opens in the editor.

The tabs on the left allow you to edit the Policy Reference Name, the Port Policy, and the Operation Policy separately. The wizard provides Overview, Source, and History views for each setting. See the online help.

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 Applications window 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.

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

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.

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. See 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, "password");
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.

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.

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 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.

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 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**.

Example 22-1 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.

  For more information about ojdeploy, see Deploying from the Command Line.
- If you do not use the `-nodatasources` switch, you can only set up passwords using application level credential mapping.

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.

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 Applications window, 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.

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. See Understanding WebLogic Server Deployment Plans.

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.

```
<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.

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 Applications window, 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.

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.
- Use the weblogic.PlanGenerator command-line tool to generate a deployment plan for an application that uses an EAR. See Generating a Template Deployment Plan using weblogic.PlanGenerator.

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.

Deploying Java Applications

JDeveloper supports deployment of Java 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 Connecting and Deploying Java EE Applications to Application Servers.

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 Applications window.
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 Applications window 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. Create the JAR file by deploying the profile to file (see previous section). This creates the JAR file, for example, `myapp.jar`.
8. Launch the executable JAR file from the command line

```
java -jar myapp.jar
```

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 How to Create and Edit Deployment Profiles.

To deploy an OSGi bundle in JDeveloper:

1. Select and right-click the project in the Applications window.
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.

Deploying Java EE Applications

JDeveloper supports deployment of Java EE applications containing a variety of technologies to a variety of application servers.

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.

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 application. 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 Applications window 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 Applications window 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, WAR, and GAR 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, Deploy to GAR, 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.

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 Applications window 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 Applications window, then choose **Deploy > deployment profile**.
3. In the Deployment Action page of the Deploy dialog, choose **Deploy to RAR file**.

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 Applications window 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.

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.

For more information, see How to Package and Deploy Customized Applications in *Developing Fusion Web Applications with Oracle Application Development Framework*.

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 Applications window, right-click the project and choose **Deploy** and select the 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 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 the Java SE browser plugin or bundle the Swing libraries for JVMs version 1.1 with your applet.

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. See Creating Shared Java EE Libraries and Optional Packages.

To create and deploy a shared library archive:

1. Create a shared library deployment profile, (Shared Library JAR file).
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 Applications window 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 **Finish**.

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.

Post-Deployment Configuration

You can migrate deployed applications from one Oracle WebLogic Server to another.

After you have deployed your application to Oracle WebLogic Server, you can migrate it.

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.

Testing the Application and Verifying Deployment

You can test deployed applications in a browser from the information in JDeveloper's log window.

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.

Deploying from the Command Line

Learn how to deploy applications from the command line.

JDeveloper deployment is built around a Design-Time data structure called a profile. A common implementation is an ArchiveProfile that describes the structure of a JAR archive. Deployment profiles can be created as part of a JDeveloper project or workspace. JDeveloper a offers a command-line tool, ojdeploy that allows deployment of ArchiveProfile(s) without invoking the JDeveloper GUI. This is the simplest form of deployment. It has the following characteristics:

`ojdeploy` can run a deployment locally in-process, or submit to a background server, ojserver. For more information, see Using ojserver.

Before deploying from the command line, you need to run JDeveloper at least once to create a deployment profile for either the application or the project.

Deployment profiles are stored as part of either the application or project properties.

ojdeploy

`ojdeploy` is available from the command line at *`jdeveloper_install`*`/jdeveloper/jdev/bin`, and the usage is

```
ojdeploy <commandId>
```

For additional details, type:

```
ojdeploy <commandId> -help
```

Currently the only available commands is `deployToArchive` (the default). This command deploys to an Archive File.

Usage:

```
ojdeploy -profile <name> -workspace <jws> [ -project <name> ] [ <options> ]
ojdeploy -buildfile <ojbuild.xml> [ <options> ]
ojdeploy -buildfileschema
```

You can use the following arguments with ojdeploy.

Table 22-2 Arguments That Can be Used With ojdeploy

Argument	Description
-buildfile	Full path to a build file for batch deploy.

Table 22-2 (Cont.) Arguments That Can be Used With ojdeploy

Argument	Description
-buildfileschema	Print XML Schema for the build file.
-profile	The name of the Profile to deploy. 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.
-project	Name of the JDeveloper project within the `.jws` where the profile can be found. If omitted, the profile is assumed to be in the Workspace.
-workspace	Full path to the JDeveloper workspace file(`.jws`).

You can use the following options with ojdeploy.

Table 22-3 Options Available to Use with ojdeploy

Option	Description
-address	The listen address for `ojserver` if not using default (`localhost:2010`). The default parts of the address may be omitted, for example, `-address :2001` or `-address fasup-pls01`
-basedir	Lets Workspace path names be interpreted relatively. The built-in macro `${`*base.dir*`}` captures the value of `-basedir`.
-clean	Clean output directories before compiling
-datasources	Deprecated For JEE applications, the `-nodatasources` option prevents the `data-sources.xml` file from being updated with connection information found in the IDE. This is ignored for non-JEE applications
-define	Define variables as comma separated name=value pairs
-failonwarning	Prevents the build system from being invoked. This is useful if a workspace or project just needs to be packaged, and not compiled at this time. Adding `-clean` will delete all files from the project output directory before compiling. Deployment will stop for that profile, if a file or directory could not be deleted.
-forcerewrite	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 a Workspace or Project just needs to be packaged, and not compiled at this time. Adding `-clean` will delete all files from the Project output directory before compiling. Deployment will stop for that profile, if a file or directory could not be deleted.
-nodatasources	Does not include datasources from IDE.
-nodependents	Tells `ojdeploy` not to navigate to project and profile dependencies. This means dependent projects will not be automatically compiled and dependent profiles will not be automatically deployed. This is useful in situations where the dependencies between projects and profiles are being calculated externally or when multiple instances of `ojdeploy`(s) are working on the same set of projects and workspaces.
-ojserver	Runs the deployment job on an ojserver. All paths referenced by the other options should be accessible on the server.
-outputfile	Redirects any JAR files created from the profile. The default is within a `\deploy` directory inside the Project or Workspace. Automatic file extension: If the `-outputfile` parameter does not specify a file extension, `ojdeploy` will figure out the extension by looking at the original file name in the profile.

Table 22-3 (Cont.) Options Available to Use with ojdeploy

Option	Description
-project	If specified, tells `ojdeploy` to look for the profile in the project. Specify the name only, without the path or `.jpr` extension
-statuslogfile	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.
-stderr	Lets these respective streams be redirected to a file for each profile and project. Macros are allowed in the name or path of the files.
-stdout	Redirect stdout to file
-timeout	Lets the user specify the number of seconds after which deployment of a single profile should be aborted. If the profile is dependent on other profiles that also need to be deployed, they all have to complete within this time.
-updatewebxmlejbrefs	Updates EJB references in web.xml
-workspace	Deployment profiles can be classified into two broad categories, those that are defined at the Application or Workspace level, and those defined at the Project level. To deploy a Workspace-profile, `ojdeploy` takes the workspace location, and the name of the profile. To deploy a Project-profile, it takes an additional `-project` argument.

You can use the following built-in Macros with ojdeploy.

Table 22-4 Built-in Macros Available to Use with ojdeploy

Macro	Description
${workspace.name}	Name of workspace (excluding `.jws`)
${workspace.dir}	Directory containing the `.jws` file
${project.name}	Name of project (excluding `.jpr`)
${project.dir}	Directory containing the `.jpr` file
${profile.name}	Defined name of the profile
${deploy.dir}	Default deploy directory (usually `${`*project.dir*`}/deploy` or `${`*workspace.dir*`}/deploy` for project-level and workspace-level profiles respectively)
${base.dir}	Value of the `-basedir` parameter, or current directory.

> **Note:**
>
> ${project.name} and ${project.dir} 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 a workspace-level profile:

```
ojdeploy -profile earprofile1 -workspace /usr/jdoe/Application1/Application1.jws
```

Deploy all profiles from all projects of a workspace:

```
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
```

Using ojdeploy from Mac OS X Platforms

If you are using ojdeploy on a on Mac OS X platform, you must set the variable `SetSkipJ2SDKCheck` in the file *`jdev_install`*`/jdeveloper/jdev/bin/jdev.conf` to true. Your entry should look like this:

```
SetSkipJ2SDKCheck true
```

Using ojdeploy

JDeveloper currently supports deployment in Ant scripts. See How to Deploy from the Command Line Using Ant .

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.

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 adds 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, for example:

```
ojdeploy -buildfile /home/jdoe/ojbuild.xml -nocompile -nodatasources
```

This adds the `-nocompile` and `-nodatasources` parameters to the default section of the build file.

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 am application, 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>
```

How to Use Wildcard Samples

Project and profile names can be specified as "*" or "name*" or "name1,name2,name3,..." or any combination of these. Application names need to be enumerated, so "*" is not allowed in application names, but application names can be specified as "application1" or "application1,application2,application3".

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>
```

How to Create a Log File for Batch Deployment

The `-statuslogfile` parameter lets the user create a build summary for the entire deployment batch. An absolute path should be specified without 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.

The following is an 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(ModulePackagerImpl.jav
a: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(ModulePackagerImpl.jav
a: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>
```

Timeouts

The `-timeout` parameter can be used for terminating runaway deployments. It specifies an upper limit on how much time deployment can take.

Timeout is for each Profile, not for the entire batch.

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 application. With `ojdeploy` you can deploy 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 must be created manually. The resulting deployed archive depends on the version of JDeveloper used, and which extensions are enabled when command line deployment is invoked.

> **Note:**
>
> 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 dependencies for the classes and JAR files to be resolved manually.

The `ojdeploy` task is an extension of the Ant `<exec>` task, so any valid attributes of that task can be used with `ojdeploy`, such as`arg` and `failonerror`.

For example, to have a deployment stop when it reaches an error, add `failonerror="true"`.

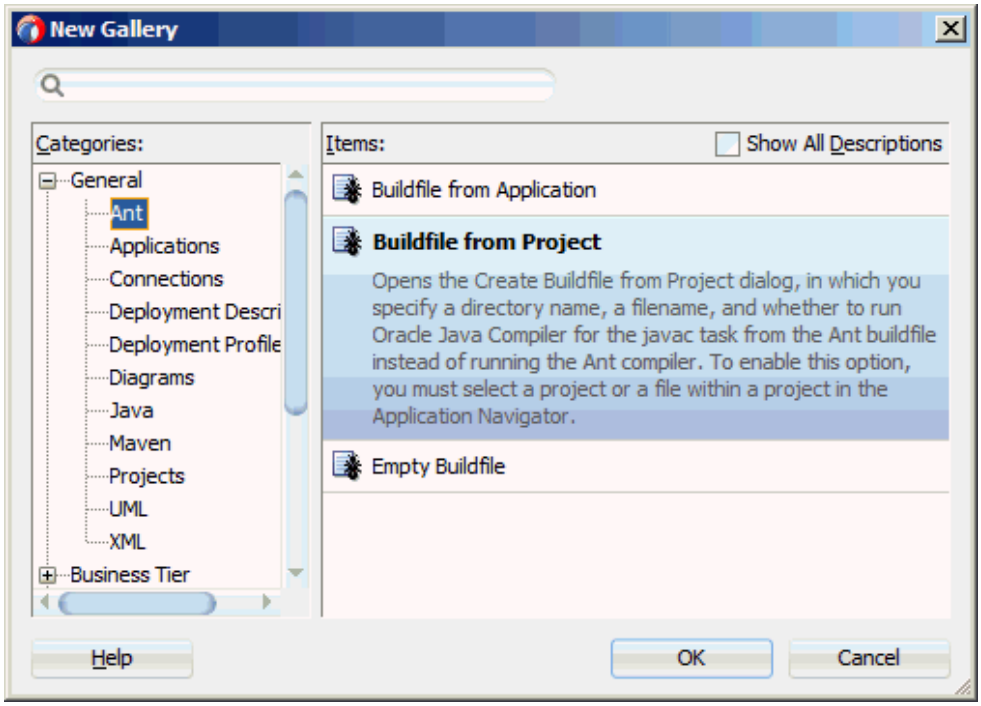
For more information see the Exec tasks section of the Apache Ant Manual at http://ant.apache.org/manual.

How to Generate an Ant Build Script

To make it easier to create an Ant build script for command line deployment, you can generate an Ant script from JDeveloper.

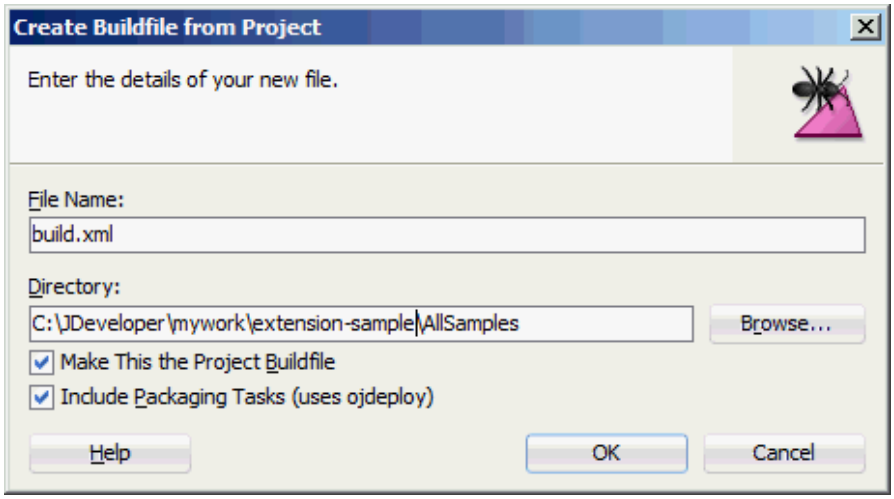
1. In the Projects window select a project or an application, and select **File > New > From Gallery**. In the General category, choose Ant. In the Items pane select the Buildfile type, then click **OK**.

Figure 22-4 Selecting Buildfile Type



2. Complete the Create Buildfile from Project dialog and click **OK**. To invoke ojdeploy from this script enable Include Packaging Tasks.

Figure 22-5 Provide File Details



The generated file has the structure shown in the build.properties file in About The build.properties File.

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.

The following is an example of build.xml.

```
#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
```

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 Applications window and selecting the "all" or the "deploy" targets.

> **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.

The following is an 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
```

ojdeploy for Extension Developers

The `ojdeploy` command offers a framework that allowed extensions to plug into the ojdeploy process. Prior to the new framework, `ojdeploy` only allowed deployment to archives. The new framework is designed to handle the creation and execution of any DeployCommand implementation. The framework allows, for example, any extension to define `ojdeploy` support for its DeployCommand.

The new framework allows extensions the freedom to:

- Define their own OJDeploy arguments and parsing rules.
- Define their own Command Parser and Model (optional).
- Define their own argument expansion and wildcard logic via ContextIterators. (optional)
- Control the exact makeup of their DeployCommand Context.
- Control the creation and setup of their DeployCommand instances.

The framework is designed to allow a great deal of flexibility while offering extension writers a mechanism that is easy to use because it provides many built-in components; for example, AbstractCommandSupport and AbstractContextIterator.

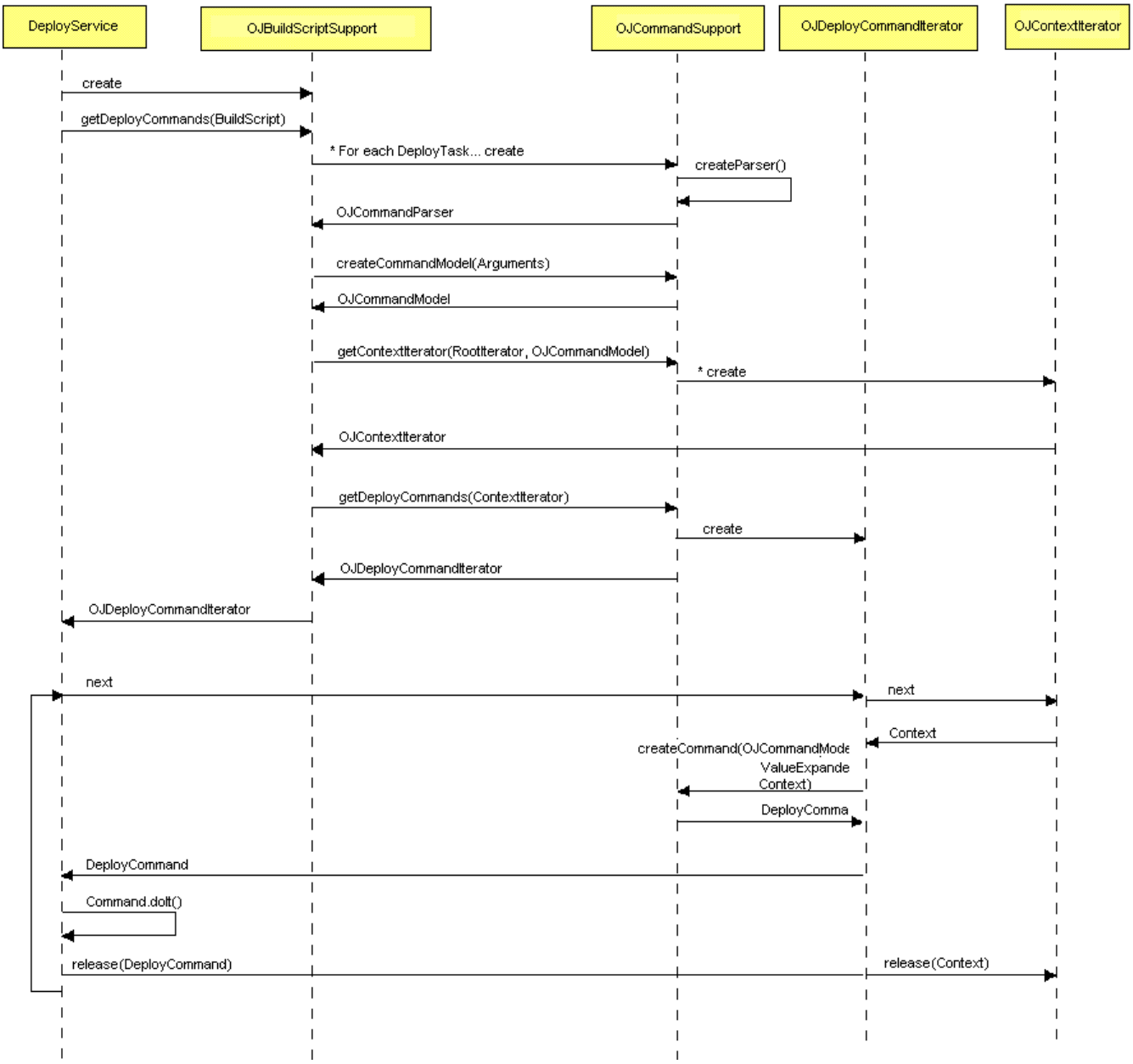
Table 22-5 Class Lexicon

Class Name	Description
Arguments	Represents command line arguments passed into the ojdeploy process. Note that Buildfiles parameters/variables are also converted into Arguments for processing.
BuildScript	A BuildScript represents the buildfile that is the input of the OJDeploy process. Note that a BuildScript can be created from a buildfile or by parsing command line arguments. Regardless, a BuildScript is always created during the OJDeploy process.
OJBuildScriptSupport	This class converts the BuildScript into an ojdeploycommanditerator. As shown in the sequence diagram below, this class controls the OJCommandSupport creation, parsing of arguments and the creation of OJContextIterators.
OJCommandSupport	This class is used to support the proper creation of a given DeployCommand. For example, the DeployToArchiveCommandSupport is one implementation that allows Deployment of Profiles to an archive file. This is where the bulk of the logic will be implemented for your own `ojdeploy` command extension. You can refer to the following HOW To section for instructions on How to write your own OJCommandSupport implementation.
Context	An IDE Context. The context is critical because it is used in order to create any DeployCommand instance via the DeployCommandFactory. As an ojdeploy extension, you will be required to construct the Context necessary to instantiate your DeployCommand instances.
OJContextIterator	An Iterator of Contexts. Note that various Context Iterators have been pre-built for you to use. For example, the ojdeploy framework includes iterators that will populate Context instances based on a given Workspace (WorkspaceContextIterator), Project (ProjectContextIterator) and Profile (ProfileContextIterator). You likely will not need to create your own ContextIterator but in the event that you do, the framework also includes an AbstractContextIterator that you can extend for that purpose (more on this in the following section on How to write your own OJContextIterator.).
DeployCommand	The entire purpose of the ojdeploy process is to create DeployCommand instances and run them. A DeployCommand is an extension of the IDE Command.

Table 22-5 (Cont.) Class Lexicon

Class Name	Description
DeployService	The DeployService is invoked by the ojdeploy process to control creation and execution of a DeployCommand in the ojdeploy process. It is in effect the client of the OJDeploy Framework API as shown in the sequence diagram below.
OJCommandModel	An OJCommandModel is an extension to the IDE CommandModel. It holds the arguments parsed into the OJCommandParser.
OJCommandParser	The OJCommandParser is used to parse Arguments and create the OJCommandModel.
OJCommand	Not to be confused with the DeployCommand, the OJCommand is a simple class which simply holds the CommandID (for example, "Deployment.DeployToFile") and Command Label (DeployToArchive). The CommandID is used by the DeployCommandFactory in order to create the DeployCommand. The Command Label is what is used as the DeployCommand identifier on the command line or buildFile. The Command Label is simply a user friendly representation that corresponds to the CommandID. There is therefore a one to one relationship between the two.
OJDeployCommandIterator	The OJDeployCommandIterator is used to iterate all DeployCommand instances of a given BuildScript. Note that the OJDeployCommandIterator in conjunction with the OJContextIterators allow for lazy loading of all DeployCommand context objects (Workspace, Project, Profile etc) and also releases those objects automatically after each command execution for proper resource management during the ojdeploy process.

Figure 22-6 Sequence Diagram



Current Limitations

- To recreate a deployed module with ojdeploy just as if it were being deployed from within the IDE in GUI mode, the exact same JDeveloper installation should be used. A change in the Jdeveloper installation means a possible change in the extensions that are loaded, or their versions, and thus the end result may also change.
- Startup time of `ojdeploy` is dependant on startup time for a headless JDeveloper. All the extensions that are registered for this mode, need to be initialized. This time could be shortened by each Extension optimizing itself for the headless mode. For large builds, the `-ojserver` option can submit a deploy job to a server running in the background. See OJServer

Using the headless Attribute

`ojdeploy` runs JDeveloper in headless (non-GUI) mode, which enables all the various extensions that support this mode.

- To ensure IDE Addins (`oracle.ide.Addin`) are loaded when running in headless mode they should be marked with the "`headless=true`" tag. Addins that are not marked as such, should not be referred to from these headless Addins.

  During `Addin.initialize()` all the IDE contexts are available, including the current Active Workspace and Project.
- All Deployment Toolkits provided by the extension that need to run in ojdeploy also need to be marked with the "`headless=true`" attribute

Using ojserver

`ojserver` is a headless version of JDeveloper that runs in server mode, listening to remote client requests. Clients and submit service requests to the server using RMI. ojserver's main purpose is to reduce the startup time incurred when using command-line tools in batch scripts that run JDeveloper in headless mode. ojserver can be found in the `oracle/jdeveloper/jdev/bin` directory.

To see the usage, parameters, options, and examples for ojserver, at the command line for *`jdeveloper_install`*`/jdeveloper/jdev/bin` type `ojserver`.

`ojdeploy` can run a deployment locally in-process, or submit to a background OJServer using the `-ojserver` option. When using `-ojserver`, `ojdeploy` blocks till the deployment is complete, and exit codes are set depending on how deployment fared on the server.

All the options for local deployment are also available when running on `ojserver`, however, if any absolute paths are being used, they have to be accessible on the server.

Deploying Using Java Web Start

Learn how to use Java Web Start to deploy Java applications so that they can be launched from an internet browser.

Java Web Start is included in the Java Runtime Environment (JRE) as part of Java SE 6 and later: http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136112.html.

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 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.

> **Note:**
>
> Starting with Java SE 7 Update 21 in April 2013 all Java Applets and Web Start Applications are encouraged to be signed with a trusted certificate. And starting with 7u25, all files must be added to JARs prior to signing. For more details, refer to the Java web site at http://www.oracle.com/technetwork/java/javase/tech/java-code-signing-1915323.html.

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.

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.

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:

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.

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.

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 Applications window, 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 Applications window 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 Applications window and choose **Open**.
10. (Optional) To reopen a project deployment profile later to make changes, right-click the project in the Applications window 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`.

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 Applications window, 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:

```
<%@ 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>
```

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 Applications window. 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 Applications window and choose **Open**.
7. (Optional) To reopen a project deployment profile later to make changes, right-click the project in the Applications window 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`.

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.

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. See https://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 How to Deploy WebLogic SCA Applications to Integrated WebLogic Server.

See WebLogic Spring SCA Applications .

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 *Developing and Administering Spring Applications for Oracle WebLogic Server*.

Installing the Weblogic SCA Spring Extension

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.

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.

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.

Example 22-2 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.

Example 22-3 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.

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 following example shows the 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 Components window, and the Properties window 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.

Components Window Features

You can select tags from the Components window 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 Components window 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 Components window and the components listed are EJB Binding and Webservice Binding.

Properties Window Features

The Properties window allows you to edit the properties of tags.

- Changes in the Properties window are synchronized with the source editor view.
- Lists of values are shown when they are relevant for a specific property.

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 Applications window right-click spring-context.xml under the project node in the Applications window 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 Servers window, you can see the services deployed under the Web Services and EJB nodes under IntegratedWebLogicServer.

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 *Developing and Administering Spring Applications for 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.

Example 22-4 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 Servers window, you can see the services deployed under the Web Services and EJB nodes under the connection node for the application server.

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

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.

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 Applications window, 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.

Troubleshooting Deployment

There a number of common problems that you may come across when deploying applications. Learn what they are and see suggestions for solving them.

This topic 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.

Common Deployment Issues

This section contains information about issues that may arise when deploying to both Integrated WebLogic Server and Oracle WebLogic Server.

[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.

How to Troubleshoot Deployment to Integrated Application Servers

This section contains information about issues that are specific to running on integrated application servers.

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 (that is, not orphaned) by pressing the **Terminate button** twice.

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

`setStartupEnv.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.

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, described in Setting the User Home Directory.
- Delete the old system directory, so that JDeveloper regenerates a new system directory.
- In the Application Servers window, right-click on **IntegratedWebLogicServer** and select **Delete Default Domain**.

How to Troubleshoot Deployment to Oracle WebLogic Server

This section contains information about issues that are specific to deploying to Oracle WebLogic Server.

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.

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.

[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.

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`.

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
```

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.
```

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.

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.

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.

Deployment Fails When EAR Contains Spaces

WebSphere deployment on Windows does not work when the directory containing the EAR generated by JDeveloper contains spaces.

Deployment Fails When the Path to the WebSphere Server Contains Spaces

WebSphere deployment on Windows does not work when the path to `wsadmin.bat` contains spaces. See Connecting to WebSphere Server.

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.

23

Getting Started with Working with Databases

This chapter describes how to get started using JDeveloper to work with databases. This chapter includes the following sections:

- About Working with Databases
- Getting Started With Oracle Database Express Edition
- How to Manage Database Preferences and Properties

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.

To get started quickly using Oracle databases in JDeveloper, see:

- Using Oracle Database 21*c* Express Edition. See Getting Started With Oracle Database Express Edition.
- Creating connections to Oracle and non-Oracle databases. See Connecting to Databases.
- Working in the Databases window. See Using the Databases Window.
- Database Development with JDeveloper. See "Database Development" at `http://docs.oracle.com/cd/E18941_01/tutorials/toc.htm`.

Connecting to and Working with Databases

Usually you start working with a database by creating a connection to it, or by importing an existing connection. 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 Databases window, 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.

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. Alternatively, you can create an offline database and working either in the Applications window or the database modeler 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 Applications window or Databases window, where they are available to applications you are working on, or in the Resources window, 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 modeler or work with in the Applications window.

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 21*c* Express Edition (Oracle Database XE).

Getting Started With Oracle Database 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 21*c* Express Edition (Oracle Database XE).

Oracle Database XE is an entry-level, small-footprint version of Oracle Database 21*c*. It is free to develop, deploy, and distribute; fast to download; and simple to administer. Use it to start learning about using databases with JDeveloper. You can download it from Getting Started: Oracle Database 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, use the Oracle Database Express Edition and Oracle Application Express documentation to create and administer users, and 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. How?

In the Create Database Connection dialog, use the following values. Leave blank any fields that are not mentioned.

Table 23-1 Connection details for Oracle Database Express Edition

Field	Value
Create Connection In	Choose **Application resources**. The connection will be displayed in the Applications window, under Application Resources.
Connection Name	Enter a meaningful name for this connection.
Connection Type	`Oracle (JDBC)` (default).
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.

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.

The preferences are available in the Preferences dialog, available from the **Tools** menu:

Table 23-2

Node in Preferences Dialog	Values that can be set
Database	Choose not to have date and time default values validated. Set the default path for export DDL files.
Database: Advanced	Set options such as the SQL array fetch size and display options for null values.
Database: Autotrace/Explain Plan	Specify the information to be displayed on the Autotrace and Explain Plan pages in the SQL Worksheet.
Database: Drag and Drop	Specify the type of SQL statement created in the SQL Worksheet when you drag an object from the Databases window into the SQL Worksheet.
Database: Licensing	Some JDeveloper features require that licenses for specific Oracle Database options be in effect for the database connection that will use the feature. This page enables you to specify, for each defined connection, whether the database has the Oracle Change Management Pack, the Oracle Tuning Pack, and the Oracle Diagnostics Pack. For each cell in this display (combination of license and connection), the value can be true (checked box), false (cleared box), or unspecified (solid-filled box). If an option is specified as true for a connection in this pane, you will not be prompted with a message about the option being required when you use that connection for a feature that requires the option.
Database: Navigation Filter	Specify the type of SQL statement created in the SQL Worksheet when you drag an object from the Databases window into the SQL Worksheet. Control the types of objects that appear in the Databases window for connections to Oracle and third-party databases. Select **Enable Navigation Tree Filtering** to choose the tab for the database type you want. For each type you can select the types of objects to appear in the hierarchy for connections to that type of database.
Database: JDBC Driver Options	Register and manage JDBC drivers for the BI JDBC driver, and the WebLogic JDBC drivers for DB2, Informix, SQL Server and Sybase.
Database: NLS	Specify globalization support parameters, such as the language, territory, sort preference, and date format.
Database: ObjectViewer	Specify whether to freeze object viewer windows, and display options for the output.
Database: PL/SQL Compiler	Specify options for compilation of PL/SQL subprograms.

Table 23-2 (Cont.)

Node in Preferences Dialog	Values that can be set
Database: Reports	Choose that database reports in JDeveloper are closed when the database is disconnected. Select the limit for the number of rows for a chart. The default is `1,000`.
Database: SQL*Plus	Set the path to the SQL*Plus command line tool.
Database: SQL Editor Code Templates	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	Allows you to control how statements in the SQL Worksheet are formatted.
Database: User Defined Extensions	(Not used by JDeveloper.)
Database: Utilities	Provides default values for utility wizards and editors
Database: Worksheet	Specify options for the SQL Worksheet.
Diagrams: Database	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.

24

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:

- Using the Databases Window
- Using the Database Cart
- Using the Structure Window
- Using the Database Reports Window
- Using the Find Database Object Window
- Using the SQL Worksheet
- Using the SQL History Window
- Using the Snippets Window
- Using the Database Object Viewer
- Using the PL/SQL Source Editor
- Using SQL*Plus
- DBMS Output Window
- OWA Output Window

Using the Databases Window

The JDeveloper Databases window provides you with a complete editing environment for online databases. You can create, update and delete database objects.

The Databases window is integrated with:

- The SQL Worksheet. More
- The Database Object Viewer. More
- The Database Cart. More

You can drag database objects from a database connection onto a database diagram to either:

- Model the database objects on the diagram.
- Reverse engineer database objects to a project, and model the offline database objects on the diagram.

For more information about database modeling, see Modeling with Database Diagrams .

When you first open the Databases window, it appears in the default docked position, which is in the upper left-hand corner, flush with the main work area of JDeveloper. When more than one window is open in the same position, each appears with a tab displaying its name.

The top-level nodes in the Databases window are:

- IDE Connections. These are globally defined connections available for reuse, and the connections are also listed in the IDE Connections panel of the Resources window from where you can copy IDE connections to the Applications window to use them within an application. More
- Application Connections. These are connections defined for use in the applications named in the connection node.
- Cloud Connections. These are connections to Oracle Database Cloud Service instances.

Right-click on a node within the Databases window 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 24-1 Databases Window 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 Databases window 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 Databases window and wait for the connection to be established.
	Collapse All	Click to collapse all expanded nodes.

You can perform various tasks from the context menus in the Databases window. 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:

- 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.
- Delete the database connection.
- Generate SQL from database objects.
- Reverse engineer 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
- Recompile Schema
- 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.

Using the Database Cart

The Database Cart allows you to deploy Oracle Database objects from one or more database connections to an Oracle Database Service instance, or to a ZIP file.

> **Note:**
>
> You cannot use the Database Cart to work with offline database objects.

You can choose to deploy:

- Just the DDL defining database objects
- DDL defining the database objects and the associated data
- Just data, for database objects that already exist in the Oracle Database Service instance

You add objects to the cart by dragging objects from the Databases window and dropping them into the Database Cart window. Alternatively, you can use the contents of a cart that you have previously saved.

From the cart, specify any desired options, and do one of:

- Click Deploy to Cloud button to display the Deploy Objects to Cloud dialog, where you choose the connection to deploy the objects to.
- Click the Deploy button to display the Deploy Objects dialog where you specify details of the deployment file to generate.

Before you use the Database Cart, ensure that you have correctly set the preferences you want to use.

From any cart tab, you can right-click and choose:

- **Close**: Closes the current cart tab.
- **Close Others**: Closes all cart tabs except the current one.
- **Close All**: Closes all cart tabs.

- **Rename**: Renames the current cart tab (for example, if you wanted to change `Cart_1` to `HR_objects`).

Table 24-2 describes the operations you can perform from the Database Cart Window toolbar.

Table 24-2 Database Cart Window Toolbar

Element	Description
	Click to open a new empty cart.
	Click to open a saved cart by specifying the XML file that specifies the cart contents. How? If the cart currently contains any objects, you are asked if you want to remove the current objects from the cart before opening a saved cart. • **Yes** Empties the current cart and fills the cart with objects from the cart you are opening. • **No** Does not empty the current cart, but adds the objects from the cart you are opening to the current cart objects.
	Click to open the Save as Cart dialog, where you can save the contents of a cart as an XML file which you can later open and use again.
	Click to open the Save as Cart dialog, where you can save the contents of a cart as an XML file which you can later open and use again.
	Click to open a Save as Cart dialog for each currently open cart.
	Click to bring the cart to the left to the front.
	Click to bring the cart to the right to the front.
	Click to refresh the cart and validate the objects in the cart against those in the Databases window.
	Click to deploy the selected objects in the cart to an Oracle Database connection. The Deploy Objects to dialog opens where you can specify additional information.
	Click to create a deployment file. The Export Objects dialog opens where you can specify additional options, generate the deployment scripts, and optionally generate a .zip file that contains them.
	Click to open the Diff Objects dialog, which allows you to compare the selected objects with the objects in another currently open cart tab or a database connection that has access to the destination objects to be compared.
	Click to open the Copy Objects dialog, which allows you to copy the selected objects to a database connection.

Table 24-3 describes the operations you can perform from the Selected Objects toolbar.

Table 24-3 Selected Objects Toolbar

Element	Description
	Click to add an initial script to be run before the object is created. Alternatively, click the down arrow next to the icon and choose from: • Add Initial Script • Add Final Script, which is run after the object is created. Alternatively, you can open a scripts dialog by moving the cursor to the Scripts field on the row for the object, and clicking the button.
	Click to remove the selected row from the cart.

Initially database objects are shown in order of type, by owner, by name. You can reorder the rows using the shuttle buttons. Table 24-4 describes the content of the table.

Table 24-4 Selected Objects Table

Element	Description
Include	Either select or deselect objects row by row, or select or deselect in the column header for all objects. Selected objects will be included in the deployment action.
Type	The type of the database object.
Owner	The owning schema of the database object.
Name	The name of the database object.
DDL	Either select or deselect objects row by row, or select or deselect in the column header for all objects. Selected objects will be included in the DDL generated by the deployment action.
Data	Either select or deselect objects row by row, or select or deselect in the column header for all objects. Only objects that have associated data have an entry in this column, for example tables, views. Selected objects will be included in the DDL generated by the deployment action.
Where	You can add a WHERE clause. Click in the cell for the appropriate row, then click the edit button. The Data Where dialog opens where you can specify the WHERE conditions.
Connection	The connection from which the object was selected.
Scripts	Optionally specify a SQL script to be executed first in the generated master deployment script (before the other generated scripts).

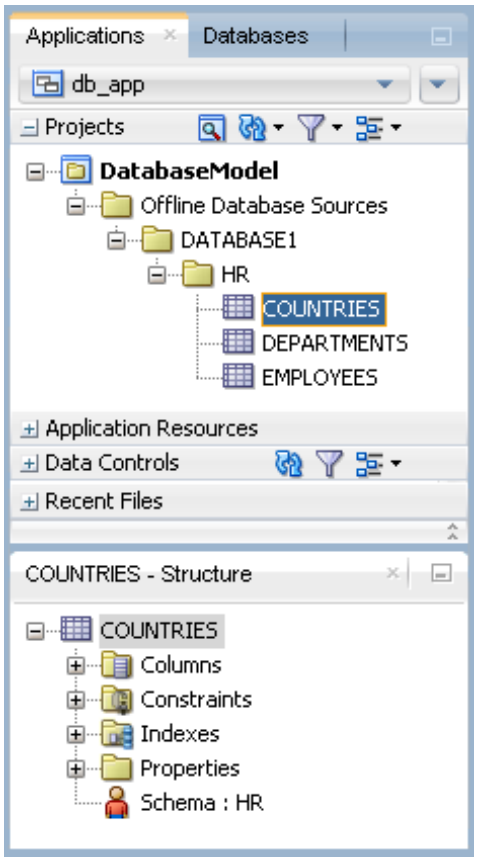
Using the Structure Window

The JDeveloper Structure window displays details of selected objects. When you are working with database objects, it displays details of the object.

The database view of the Structure window displays details of a connection or database object selected in the Databases window, or an offline database object selected in the Applications window.

For an offline database, the Structure window displays details of offline database objects such as tables, views, synonyms. When you select the object in the Applications window, details of the object are shown in the Structures window, as shown in the figure.

Figure 24-1 Structure Window View of Offline Database Object

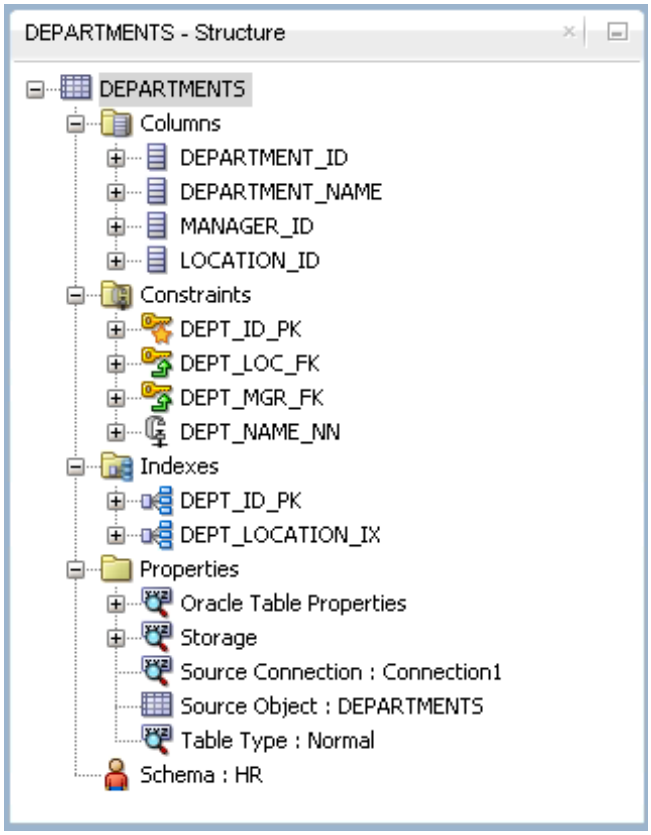


From the context menu of the offline database object, you can perform the following actions:

- Find Usages
- Use as Template
- Properties
- Go to Declaration

When you are working with database objects that have been reverse engineered from a database connection the Structure window can show you information about the original objects. For example, a table reverse engineered from an online database connection will display details of the source object and the connection used, as shown in Figure 24-2.

Figure 24-2 Reverse Engineered Object in Structure Window



When you select a database object such as a table in a database connection in the Databases window or an offline database object such as a table in an offline database in the Applications window, 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.

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.

Using the Database Reports Window

Use the Database Reports window to view pre-defined reports about the database and its objects, and to create new 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 Working with Database Reports.

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 window, or bring a currently open window to the foreground, using **Window > *window-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 24-5 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`.
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 Applications window and then clicking **Recent 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.

Using the SQL Worksheet

JDeveloper provides a SQL Worksheet that you can use to query data, by writing simple or complex SQL statements.

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.

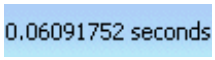
You enter SQL statements in the SQL Statement area, and use the buttons on the toolbar to perform actions.

Table 24-6 describes the icons and fields in the toolbar above the SQL Worksheet statement area.

Table 24-6 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.

Table 24-6 (Cont.) Icons in the SQL Worksheet Toolbar

Icon	Name	Function
	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. To see trace information, click the **Autotrace** tab.
	SQL Tuning Advisor (Ctrl + F12)	Opens a window that runs the SQL Tuning Advisor, which is SQL diagnostic software in the Oracle Database Tuning Pack. The Overview and Details tabs include advice or recommendations for how to tune the specified statement, along with a rationale and expected benefit.
	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.
	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.
	SQL History	Click to open the SQL History window.
		(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 **Window > Database > Databases window**.
2. Expand IDE Connections or Application Connections.
3. Right-click the connection in the window, 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. Select the **Query Builder** tab. For more information at any time, press F1 or click **Help** from within the SQL Worksheet.
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. The first time, you are prompted to choose the type of statement you want to use.
5. Use the SQL Worksheet buttons to perform the action you want, for example to run a query.

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.

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

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.

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 **SQL History** button.
   - Right-click in the SQL Worksheet to open the context menu, and select **History**.

   A window showing the list of the statements previously entered is displayed.
2. Select the desired statement from the window.
3. Click **OK**.

The statement is displayed in the upper pane of the worksheet.

Using the SQL History Window

Commands you have executed are stored in the SQL history regardless of whether you explicitly save them. The SQL History Window allows you to access commands you have 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 window, or bring a currently open window to the foreground, by choosing it from the Window 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 24-7 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.

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 24-8 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.

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 in the Database Object Viewer 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.
- Indexes, which displays details of any indexes.
- Details, which displays details of the object.
- SQL, which displays the SQL that represents this object.

Database Object Viewer Tabs Toolbars

The specific buttons on the toolbar vary from tab to tab.

Table 24-9 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.
	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.
		Click to open the Sort dialog where you specify the columns to sort by and the sort order.

Table 24-9 (Cont.) Database Object Viewer Tabs Toolbar Icons

Icon	Name	Function
		Enter a value to reduce the number of records displayed, for example `DEPARTMENT_ID>20`.
		Click to perform one of a range of common table actions.

Using the PL/SQL Source Editor

The PL/SQL Source Editor displays PL/SQL code for database objects such as procedures and functions.

In addition to the PL/SQL-specific features of the PL/SQL Source Editor, you can also use many of 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 as described inUsing the Source Editor.

In addition, there are some PL/SQL specific features which are described below.

Table 24-10 describes the operations available from the PL/SQL Source Editor toolbar.

Table 24-10 PL/SQL Source Editor Toolbar

Element	Operation
	Keeps that subprogram's tab and information in the window when you click another object in the Databases window; a separate tab and display are created for that other object. To unfreeze, click the pin again.
	Use to toggle between write mode and read-only mode.
	Enter a string and click Enter. To choose from a list of search options, click the down arrow.
	(Database only) Performs a PL/SQL compilation of the subprogram. Click the arrow to choose from: • Compile for Debug, which performs a PL/SQL compilation of the subprogram so that it can be debugged. • Compile, which performs a PL/SQL compilation of the subprogram. The results are displayed in the Log window.

Table 24-10 (Cont.) PL/SQL Source Editor Toolbar

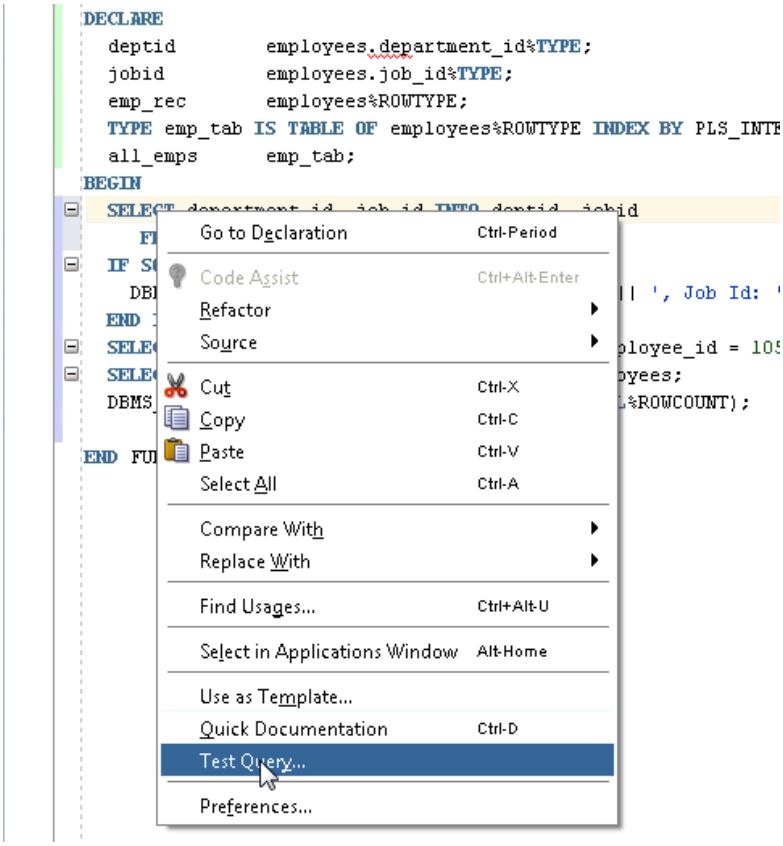
Element	Operation
	(Offline only) Performs a PL/SQL compilation of the subprogram. To enable, you must create or choose a database connection against which the subprogram will be compiled. When you click this button, the subprogram is saved to the offline database, and the program is submitted for compilation.
	(Offline only) Compile for Debug, which performs a PL/SQL compilation of the subprogram so that it can be debugged. To enable, you must create or choose a database connection against which the subprogram will be compiled. When you click this button, the subprogram is saved to the offline database, and the program along with debug information is submitted for compilation.
	Opens the Run PL/SQL window which allows you to specify arguments when running or debugging PL/SQL functions, procedures, and packages.
	Starts execution of the subprogram in debug mode, and displays the Debugging - Log tab, which includes the debugging toolbar for controlling the execution.
	Code highlight. For example, clicking on the declaration of, or any usage of `p_column_name`, and selecting this button will highlight all usages of `p_column_name`. Note that this is not a purely textual match, but is explicitly usages of the variable, obeying all scoping rules. If this button is not enabled when you have selected some code, expand the Search menu on the JDeveloper toolbar, and deselect Auto Code Highlight.
	Clear all code highlighting.
	The database connection is used for compilation. Use to choose a different database connection. Once you have chosen a connection, it is persisted and used in future instances of the PL/SQL Source Editor.

Using Test Query

For SELECT statements in PL/SQL, you can test a query against a live database to check that the correct rows are returned. Any PL/SQL variables are replaced in the test query with bind variables and clauses such as INTO are ignored.

To test the query, right-click in the SELECT statement of the PL/SQL and choose Test Query, as shown in the figure.

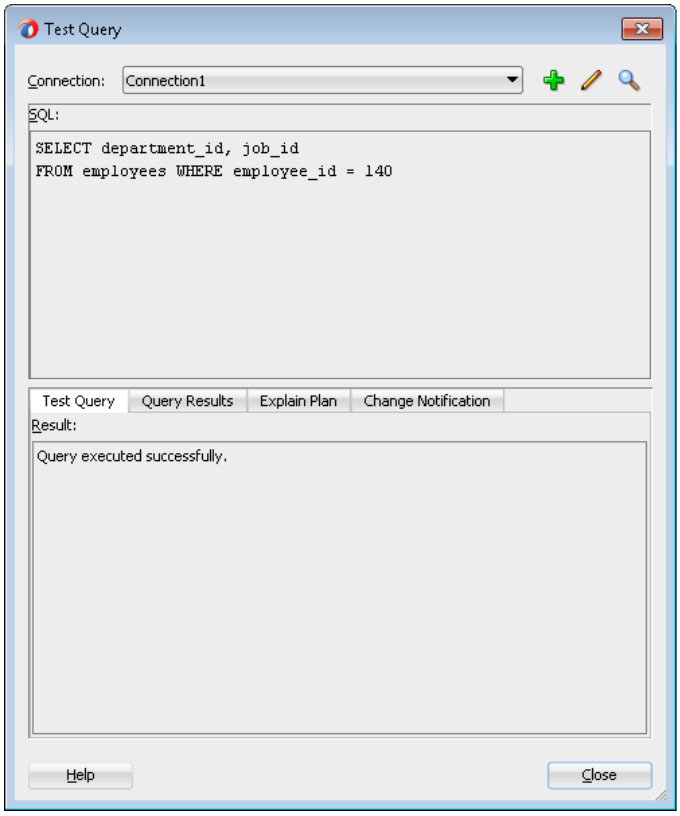
Figure 24-3 Context Menu of Select Statement



The query is run and the Test Query dialog is displayed, as shown in the figure below.

If you do not have a connection to a database defined, you must first create one. You can do this from the Test Query dialog. To choose a different database connection, choose from the list in the Test Query dialog.

Figure 24-4 Test Query Dialog

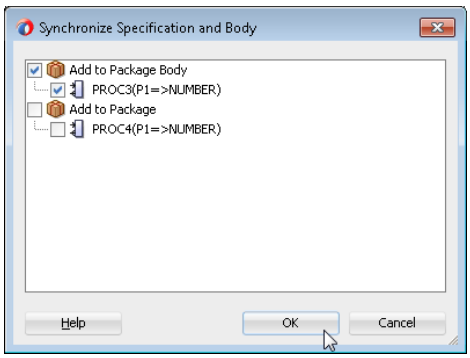


Synchronizing Package Specifications and Bodies

You can use Synchronize Specification and Body to:

- Create the initial body for a package. Right click in the specification and choose Synchronize Specification and Body from the context menu. A new package body is created.
- Identify methods that are specified in the specification which are not in the package body. Right click in the specification and choose Synchronize Specification and Body from the context menu. The Synchronize Specification and Body dialog opens, as shown in the figure.

Figure 24-5 Synchronize Specification and Body



This lists all methods that can be synchronized, and all those in the specification that are not in the body are pre-selected. Select the methods you want synchronized and click OK.

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 information about launching SQL*Plus, see *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. See *Using Scripts in SQL*Plus*.

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 https://www.oracle.com/database/technologies/instant-client.html.

You can launch SQL*Plus from:

- The Tools menu
- A database connection in the Databases window
- A SQL file in the Applications window

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 > SQL*Plus**, 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 Databases window, 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 **Window > Database > Databases window**.
2. Right-click the connection, and choose **SQL*Plus**.

To launch SQL*Plus from a SQL file:

1. In the Applications window, 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.

DBMS Output Window

The DBMS Output window is used to display the output of the buffer used to store information from a package or procedure to be read by another trigger, procedure, or package.

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.

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 24-11 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.
	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).

Table 24-11 (Cont.) DBMS Output Window Toolbar Icons

Icon	Name	Function
	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.

OWA Output Window

The OWA Output window enables you to see the HTML output of MOD_PLSQL actions that have been executed in the SQL Worksheet.

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 24-12 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.

25

Connecting to and Working with Databases

This chapter describes how to create and work with database connections.
This chapter includes the following sections:

- About Connecting to and with Working with Databases
- Configuring Database Connections
- Browsing and Searching Databases
- Connecting to Databases
- Connecting and Deploying to Oracle Database Cloud Service
- Importing and Exporting Data
- Copying_ Comparing_ and Exporting Databases
- Working with Database Reports
- Troubleshooting Database Connections

About Connecting to and with Working with Databases

From JDeveloper, you can connect to and work with Oracle databases and a number of non-Oracle databases.

Database connections can be available in the Applications window or Databases window, where they are available to applications you are working on, or in the Resources window, 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 Getting Started With Oracle Database Express Edition.

Configuring Database Connections

Learn how to connect to databases from JDeveloper.

You can define and manage connections to external data sources using the Create Database Connection dialog. Database connections are shown in:

- The Resources window, where they can be added to catalogs to facilitate collaborative working or to make them available to more than one application.
- The Databases window, where you can create, edit, and modify objects in the database.
- Application Resources panel in the Applications window, 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.

Connection Scope

In JDeveloper 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.

What Happens When You Create a Database Connection

When you create a database connection, JDeveloper creates a node for the connection in the Databases window, and an additional node in either the Resources window or in the Application Resources panel of the Applications window depending on the scope of the connection.

Team level database connections are also available when JDeveloper is configured to work with teams.

In the Applications window and Databases window, you can expand the database connection node to view and work with database objects. In the Resources window, 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 Databases window, the node for the connection is under the node with the same name as the application.

In the Applications window, 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 Applications window to use it in an application by:

- From the Resources window, dragging the connection and dropping it on the Connections node in the Applications window under Application Resources.
- From the Resources window, right-clicking the connection and choosing Add to Application.
- In the Databases window, dragging the connection 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`.

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.

How to Create Database Connections

To create a database connection:

1. If necessary load database features by choosing **Window > Database**.
2. Open the Databases window by choosing **Window > Database > Databases**.
3. Right-click **IDE Connections** or the 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**.

   Then, 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 dialog.
4. 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.

After you have defined a connection, you can return to the dialog and edit its attributes, however you cannot change the connection type after the database connection has been created.

Create a CloudDB/ATPDB Database Connection

Create a CloudDB/ATPDB connection to an Oracle Cloud Database—such as Autonomous Transaction Processing (ATP) or Autonomous Data Warehouse (ADW)—using Oracle Cloud Wallet.

Before you begin, download the wallet file from the exiting cloud database connection and save it in your local directory. You'll also need the database user name and password when creating the connection.

To create a database connection using Oracle Cloud Wallet:

1. From the JDeveloper window, click **Window**, **Database**, then **Databases** to open the **Databases** tab.

   The **Databases** tab is shown in the Explorer panel on the top left of the JDeveloper window.

2. From the **Connection** menu, click **New Connection** ( ) to open the **Create Database Connection** dialog.
3. Create the database connection:
   a. To define a database connections for the IDE, select **IDE Connection** from the **Create Connection In** list.

      If you want to create the connection for a specific application, select the application from the menu instead.

   b. Type a name for your connection in the **Connection Name** field.
   c. Select **Cloud DB** from the **Connection Type** menu.
   d. Enter the database user name and password in the appropriate fields.
4. To configure Oracle Cloud Wallet:
   a. Click the **Browse** button next to the **Configuration** field in the **Cloud DB Settings Details** tab, then navigate to the wallet zip file.

      All the wallet's tnsnames are available in the **Service** list.

   b. Select a tnsname such as **atpdb_high** from the **Service** list.
   c. From the **Proxy** tab, configure the proxy as required.

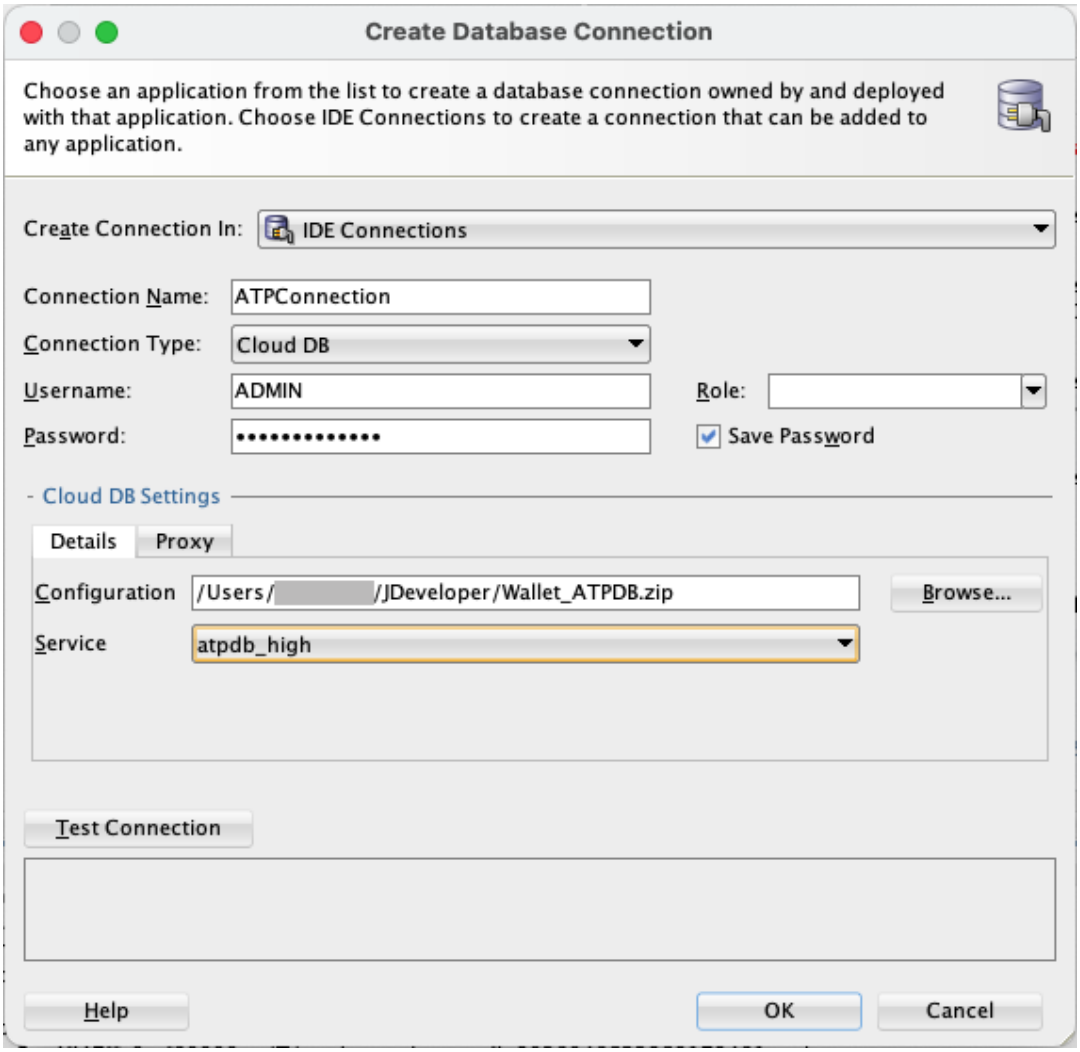
      If you are not on a VPN, select **No Proxy** (the default).

      If you are using a VPN, either select **Use jdeveloper Proxy** or select **Use Custom Proxy** and provide the required host and port.

      See Configuring Proxy Settings.

Once you've finished configuring your connection, the Create Database Connection dialog should look something like this:

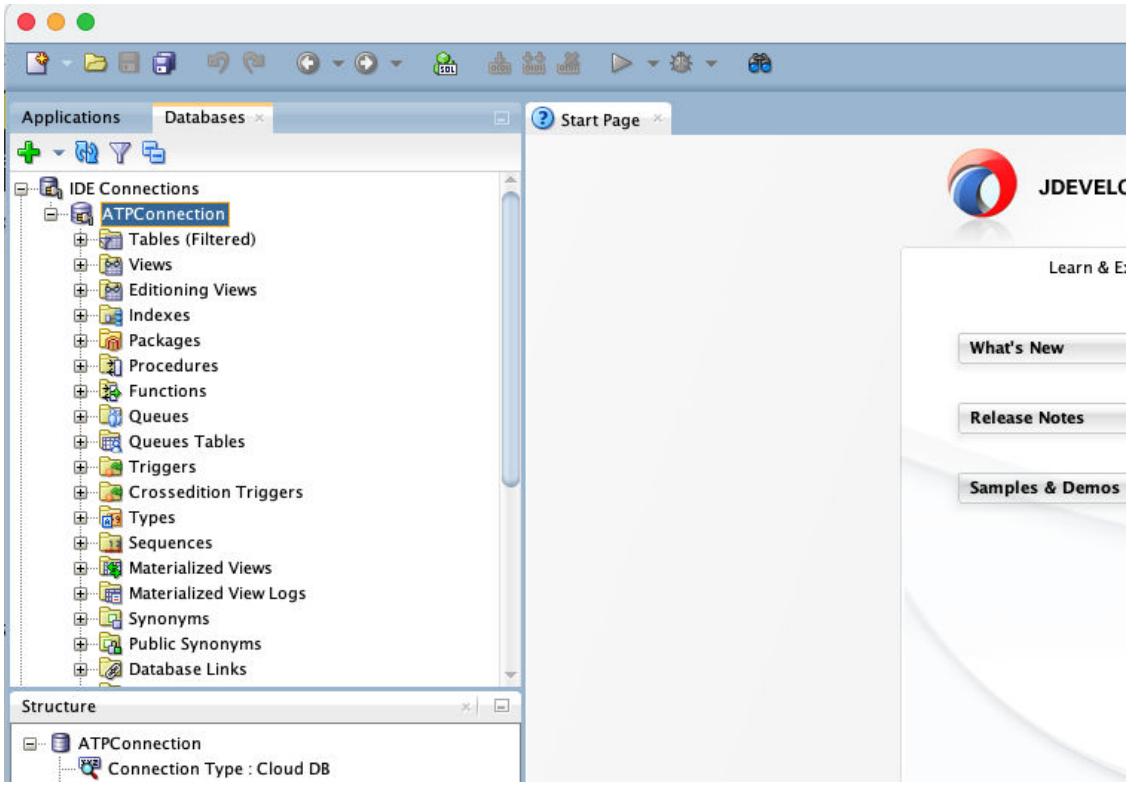
Figure 25-1 Create Database Connection Dialog for Oracle Cloud Wallet



5. From the **Details** tab, click **Test Connection**.
6. If the test is successful, click **OK** to create the connection.

You can view details for the new connection from the **Databases** tab in the left navigation pane.

Figure 25-2 Explorer Databases Tab with New Connection



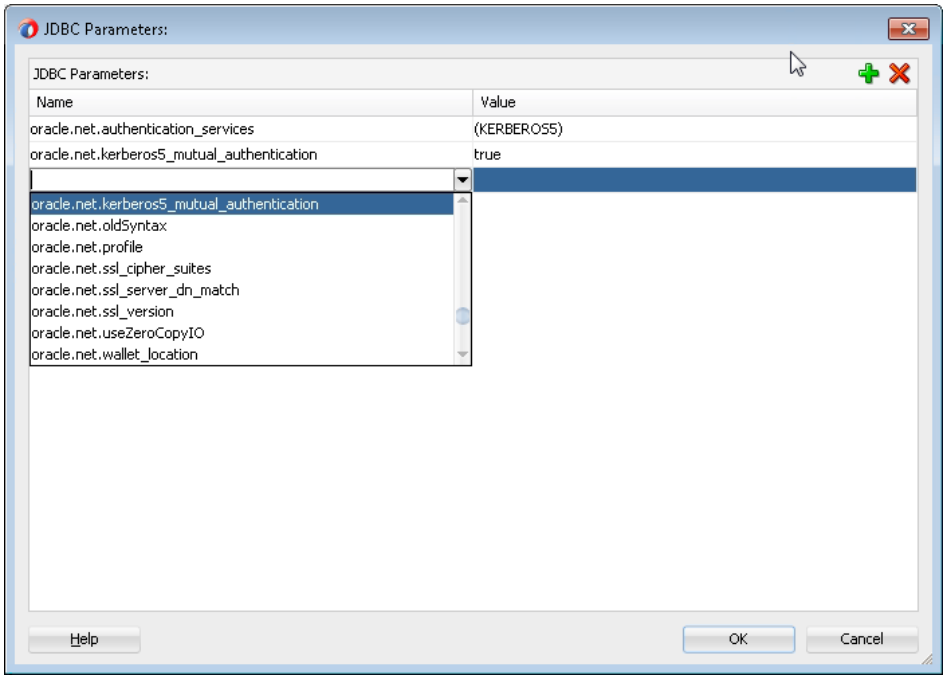
Defining Additional JDBC Parameters

If you need to use additional parameters for the JDBC connection, for example, for JDBC encryption support, you can enter them in the same connection dialog.

To define additional parameters:

1. In the Create or Edit Database Connection dialog, click **JDBC Parameters** to open the JDBC Parameters dialog.
2. The dialog allows you to enter any property name/value pairs you require in order to create the connection. For more information at any time, press F1 or click **Help** from within the dialog.
3. To add a new parameter Click . A new row is displayed. Where the driver supports it there may be a list of possible parameters that you can select from, as shown in the figure.

Figure 25-3 JDBC Parameters Dialog



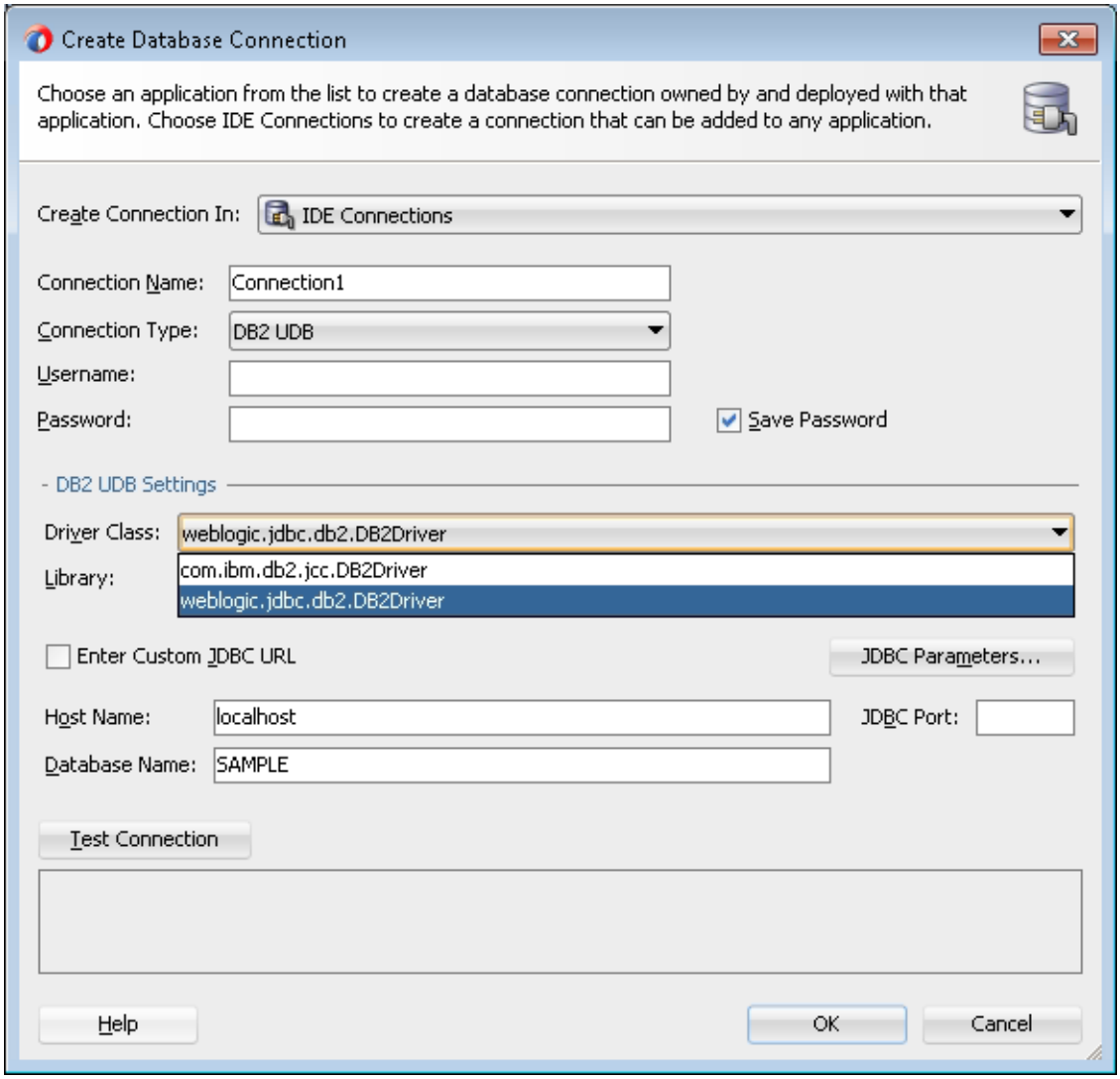
Choose or enter the parameter you want to use and enter a default value for it.

4. Click **OK** on the dialog, then **OK** on the Database Connection dialog.

Using Different Drivers

For some types of database connection, you can choose from more than one available driver. For example, if you choose DB2 UDB, as shown in the figure you can choose either of the available drivers.

Figure 25-4 Drivers Available



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 `jdbc.library`. For example:

  ```
  jdev -J-Djdbc.library=/jdev_install/jdeveloper/ojdbc6.jar
  ```

How to Edit Database Connections

To edit a database connection:

1. Choose **Window > Database > Databases**.
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.

How to Export and import Database Connections

You can import and export database connections, described below.

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 **Window > Database > Databases**.
2. Right-click either **IDE Connections** or **application** and choose **Export Connections**.
3. In the Select Connections page of the Export Connection wizard, select the connections you want to export details of, and click **Next**. For more information at any time, press F1 or click **Help** from within the wizard.
4. In the Destination File page of the wizard, specify a fully qualified filename for the file to be generated. The generated content of the file is XML, and you must use a filename with the suffix `xml`. Click **Next**.
5. If the connections contain passwords, you can encrypt the passwords in the generated file in the Password Handling page of the wizard. If you choose not to encrypt the passwords, they are not included in the generated file. Click **Next**.
6. The summary page of the wizard displays details of the export connections file. Click **Finish**.

   The connection information for the selected connections is saved in the file and can be imported for use by others.

An alternative way of exporting connections, including database connections that are IDE Connections, is to use the Resources window. For more information, see Importing and Exporting Catalogs and Connections.

Importing Database Connections

You can import connection descriptors that have previously been exported.

To import a database connection:

1. Choose **Window > Database > Databases**.
2. Right-click either IDE Connections or application and choose **Import Connections**.
3. In the Source File page of the Import Connection wizard, enter the file name of your exported connection file or click Browse to locate it, and click **Next**. For more information at any time, press F1 or click **Help**.

4. On the Password Handling page of the wizard, if a password has been used to encrypt the connection passwords in the export file, enter it here. Otherwise, select Remove all passwords from the exported connections. Click **Next**.
5. On the Select Connections page of the wizard, choose the connections that you want to import information for. Click **Next**.
6. The Summary Page summarizes information about the connections will be imported. Click **Finish**.

An alternative way of importing connections that can include database connections is to use the Resources window. For more information, see Importing and Exporting Catalogs and Connections.

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 **Window > Database > Databases**.
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 **Window > Database > Databases**.
2. Expand **IDE Connections** or **application**, and select a database connection.
3. Right-click the connection and choose **Disconnect**.

How to Delete Database Connections

Deleting connections removes them from the Databases window 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 application or project. It is best to use caution when deleting connections.

To delete a database connection:

1. Choose **Window > Database > Databases**.
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**.

How to Register a New Third-Party JDBC Driver

If you plan to use a third-party JDBC driver 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.

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.

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`.

Browsing and Searching Databases

Learn about browsing database objects, searching, and filtering the results.

You can control how much of the data source you view and how you view it, and search for database objects.

Browsing Databases

You can browse online databases and offline database objects.

Browsing Online Databases

You can browse online databases by opening JDBC connections accessible in the Databases window.

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 Databases window, 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.

Browsing Offline Database Objects

You can browse offline database objects using the Applications window.

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 Databases window.
- To view offline database objects, use the Applications window.

  Changes to database objects in projects (i.e. visible via Applications window) can be reconciled against a live database, but until reconciliation, no changes to the offline objects affect online databases.

To open the Applications window or Databases window:

1. Choose **Window** from the main toolbar.
2. To open:
   - The Applications window, choose **Applications**.
   - The Databases window, choose **Database > Databases**.

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 > Databases window**.
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.

How to Browse Offline Databases and Schemas

Browse offline databases and schemas in the Applications window to find objects such as offline tables or views.

To view offline schemas and the objects they contain:

1. In the Applications window, expand the project containing your offline schemas.
2. Expand Offline Database Sources and then expand the database and schema you want to browse.

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.

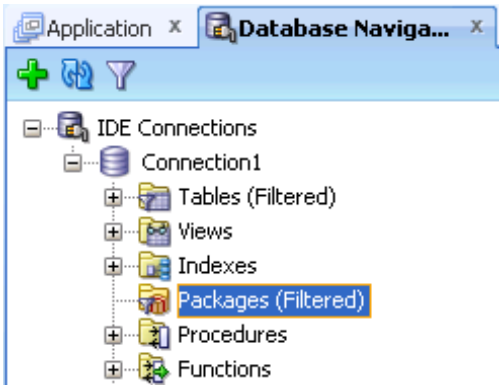
> **Note:**
>
> When you create a connection to Oracle Database, objects for the schema named in the connection are shown. To see the contents of other schemas, expand the Other Users node and then expand the node for the schema you want.
>
> If you connect via Generic JDBC or JDBC-ODBC connections, 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).

To use filters:

1. Choose **Window > Database > Databases**.
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 the figure.

Figure 25-5 Filtered Objects in Databases Window



With the object still selected, click the **Apply Filter button** in the Databases window toolbar and a dialog appears, showing the current selection, if any. From this dialog, you can change the filter currently applied.

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 **Window > Database > Databases**.
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 Databases window toolbar, click the **Apply Filter button** in the Databases window 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.

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 Databases window 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.

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 Databases window.
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.

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 **Window > 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 window.
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.

Connecting to Databases

Learn how to create connections to Oracle databases and non-Oracle databases.

This section describes how to connect to Oracle and non-Oracle databases.

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:

- Databases window.
- Applications window.
- Resources window.

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.

How to Create a Connection to Oracle Database

JDeveloper allows you to connect to a number of Oracle and non-Oracle databases.

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.

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. 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`

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`

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.

How to Create a Connection to Java DB/Apache Derby

Apache Derby is an open source relational database implemented entirely in Java. Java DB is Oracle's supported distribution of the Apache Derby open source database. 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. For more information about Java DB, see http://www.oracle.com/technetwork/java/javadb/overview/index.html.

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:

- 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`

How to Create a Connection to IBM DB2 Universal Database

JDeveloper allows you to connect to IBM DB2 Universal Database 10.1 or 9.5, or to emulate IBM DB2 Universal Database 10.1 or 9.5 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.

> **Note:**
>
> IBM DB2 Universal Database 9.5 syntax of DROP column and ALTER COLUMN is supported for IBM DB2 Universal Database 9.5.

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.
   - 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`

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`

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

Example 25-1 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.

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.

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`

Connecting and Deploying to Oracle Database Cloud Service

Learn how to work with Oracle Database Cloud Service from JDeveloper.

You can connect to and deploy Oracle Database objects to Oracle Database Cloud Service. For more information about Oracle Database Cloud Service, see https://cloud.oracle.com.

Types of JDeveloper Connection to Oracle Database Cloud Service

Create connections in JDeveloper to databases.

JDeveloper connects to an Oracle Database Cloud Service instance using two types of connection:

- Database Connection: This connection is made using the Service Home URL (identified in the "Welcome to Oracle Cloud" email). This URL connects to Oracle Application Express and you can browse database objects and deployments on your Oracle Database Cloud Service instance from a node in the Databases window. The connection is not validated until you connect to the Oracle Database Cloud Service instance, when you are prompted to provide a password.
- Secure FTP connection: The Secure FTP hostname and SFTP user identified in the "Welcome to Oracle Cloud" email are used in this connection to upload the deployment ZIP file to your Oracle Database Cloud Service instance using SFTP. At the Oracle Database Cloud Service end, there is a background job which runs to download these files and put them onto the Oracle Database Cloud Service instance.

  Once you have entered the Secure FTP hostname and user, either in the Create or Edit Cloud Connection Dialog or the Deploy Objects to Cloud Dialog, the information is saved and will be present the next time you deploy objects to your Oracle Database Cloud Service instance.

  If you have just signed up for Oracle Database Cloud Service, there are some steps you have to carry out before trying to create a connection from JDeveloper. For example, the Identity Domain Administrator has to create new passwords for the Service Home and for the Secure FTP Site. The information you need is in the "Welcome to Oracle Cloud" email.

  Also, you have to create users who can create database connections to Oracle Database Cloud Service instances. See About Database Cloud Service Roles and Users.

Creating an Oracle Database Cloud Service Connection

After you sign up for the Oracle Database Cloud Service, you will receive the connection information you need.

To create a connection:

1. If necessary, open the Databases window by choosing **Window > Database > Databases**.
2. Either click the New Connection button, or right-click the Cloud Connections node and choose **New Cloud Connection**.
3. In the New Cloud Connection dialog, enter the following information. For more help at any time, press F1 or click **Help**.
   - A name for the connection
   - The username of a user able to make a connection to Oracle Database Cloud
   - The Service Home URL
   - The Secure FTP username
   - The Secure FTP hostname

The connection is created and listed in the Databases window, but the information is not validated until you connect to Oracle Database Cloud Service.

Editing an Oracle Database Cloud Service Connection

You can only edit a connection when it is disconnected.

To create a connection:

1. If necessary, open the Databases window by choosing **Window > Database > Databases**.
2. Right-click the node for the Cloud connection under the Cloud Connections node and choose **Properties**.
3. In the Edit Cloud Connection dialog, make the changes that you want. For more help at any time, press F1 or click **Help**.

Connecting and Disconnecting from an Oracle Database Cloud Service Connection

To connect:

1. If necessary, open the Databases window by choosing **Window > Database > Databases**.
2. Expand the Cloud Connections node and either:
   - Click on + next to the Cloud connection node.
   - Or right-click the Cloud connection node and choose Connect.
3. In the Authentication dialog enter the password for the Cloud connection user. For more help at any time, press F1 or click **Help**.

The Cloud connection node expands to show the database objects present in the Oracle Database Cloud Service instance, and below those nodes previous deployments to the Oracle Database Cloud Service instance.

If an error message is displayed, check the connection properties.

To disconnect:

- Expand the Cloud Connections node, right-click on the Cloud connection node and choose **Disconnect**.

Using the Database Cart

The Cart is a convenient tool for collecting Oracle Database objects from one or more database connections, and deploying, exporting, comparing, or copying those objects. You can put objects into one or more carts, each with its own tab. When the Cart window is opened, it contains an empty cart, although you can create new carts and open previously saved carts in new or existing cart tab.

Objects in the Cart are not automatically synchronized with database activity; to update the contents of the cart with the current state of the database, click the Refresh icon. If an object does not exist after a refresh, the object is disabled in the Cart and is underlined to indicate the error

You can put database objects into a cart tab in several ways:

- Drag and drop objects from the Databases window into the Cart window.
- Select one or more objects in the Databases window, right-click, and select **Add to Cart.**
- Open a previously saved Cart XML file.
- Add scripts By using the Scripts icon drop-down (Add Initial Script, Add Final Script). (If you use a Cart tool that does not support scripts, they are ignored.)

Setting the Default Directories Used By the Database Cart

You can change the default directly used by the Database Cart, and the default directory used by scripts.

To set the cart directory:

1. Choose **Tools > Preferences**.
2. From the Preferences page, navigate to the Database: Utilities: Cart page and change the directory for the Default Cart or the Open Script directory. For more information at any time, press F1 or click **Help** from within the dialog.

Configuring Database Cart Tools

For each of the Cart database utilities (Export, Diff [compare], Copy), you can create, save, and open utility-specific configuration settings.

You can change the default locations used by the cart in the Database: Utilities: Cart page of the Preferences dialog (available from the Tools menu).

To create and save configuration settings:

1. From the database cart dialog (either Export, Database Diff, or Copy), click the Save Configuration button.
2. In the Save Tool Configuration dialog, the default location for the configuration file is shown. If necessary, browser to a different location.
3. Click **Apply**.

A configuration file is created at the location you have selected with a name that reflects the tool you invoked the dialog from:

- `export_cart.xml`
- `diff_cart.xml`

- `copy_cart.xml`

To use previously saved configuration settings:

1. From the database cart dialog (either Export, Database Diff, or Copy, click the Open Configuration button.
2. In the Open Tool Configuration dialog, the appropriate configuration file at the default location is displayed:
   - `export_cart.xml`
   - `diff_cart.xml`
   - `copy_cart.xml`

   If necessary, browse to a different location and click **Apply**.

Deploying to Oracle Database Cloud Service

This section describes how to deploy to the Oracle Database Cloud Service. As well as just deploying database objects, you can specify a script to run before and another to run after the generated scripts.

To deploy database objects to Cloud:

1. If necessary, create a Cloud connection.
2. If necessary, set database cart preferences appropriately.
3. If necessary, open the Database Cart. Choose **Window > Database > Database Cart**. For more information at any time, press F1 or click **Help** from within any of the windows or dialogs.
4. If necessary, open the Databases window by choosing **Window > Database > Databases**.
5. In the Databases window, expand the database connection or connections containing the database objects you want to add to the cart. Select the objects (hold down Ctrl to select more than one at a time) and drag them to the cart.

   You can only use database objects in database connections. You cannot use offline database objects in a Cloud connection.
6. In the Database Cart choose which objects you want to deploy, and if appropriate the data you want to deploy too.
7. In the Database Cart, click Deploy to Cloud and in the Deploy Objects to Cloud dialog choose the options you want, then click **Apply**.

   You can change the default deployment location in the Utilities: Cart page of the Preferences dialog (available from the Tools menu)
8. If the Cloud connection is not currently connected, the Authentication window appears where you enter the password for the username associated with this connection.

JDeveloper creates the ZIP file at the location you specified in the Deploy Objects to Cloud dialog, then connects to the Oracle Database Cloud Service instance by Secure FTP and transfers the ZIP file containing the deployment files.

You can examine the status of the deployment using the Log tab of the deployment object under the cloud database connection's Deployments node. After deployment, you will be able to see the deployed database objects under the Cloud Connections node in the Databases window.

Deploying to a Database Deployment File

You can create a ZIP file containing the DDL and optionally the data for database objects from one or more database connections.

JDeveloper creates a ZIP file which contains:

- A number of SQL scripts containing DDL for the database objects.
- Scripts to deploy any associated data that you want to deploy.
- A master script that has a name of the form `Generated-yyyymmddhhmmss.sql`.

Once the ZIP file has been created, you can use it to deploy the database objects to a database connection by running the master script.

To deploy database objects to a ZIP file:

1. If necessary, open the Database Cart. Choose **Window > Database > Database Cart**. For more information at any time, press F1 or click **Help** from within any of the windows or dialogs.
2. If necessary, open the Databases window by choosing **Window > Database > Databases**.
3. In the Databases window, expand the database connection or database connections containing the database objects you want to add to the cart. Select the objects (hold down Ctrl to select more than one at a time) and drag them to the cart.
4. In the Database Cart choose which objects you want to deploy, and if appropriate the data you want to deploy too.
5. In the Database Cart, click Deploy and in the Deploy Objects dialog choose the options you want, then click **Apply**.

JDeveloper creates the ZIP file at the specified location.

You can change the default deployment location in the Utilities: Cart page of the Preferences dialog (available from the Tools menu).

Saving a Database Cart

You can save the contents of the database cart to reuse later.

To save the contents of a cart:

1. If necessary, open the Database Cart. Choose **Window > Database > Database Cart**. For more information at any time, press F1 or click **Help** from within any of the windows or dialogs.
2. If necessary, open the Databases window by choosing **Window > Database > Databases**.
3. In the Databases window, expand the connection or connections containing the database objects you want to add to the cart. Select the objects (hold down Ctrl to select more than one at a time) and drag them to the cart.
4. Click Save Cart on the Database Cart toolbar and in the Save Cart dialog, check the location and click **Apply**.

JDeveloper saves the contents of the cart as an XML file at the specified location.

Opening a Saved Database Cart

You can open the saved contents of a database cart. You can either empty the cart so that it will contain just the contents of the saved cart, or add them to the existing contents of the cart.

1. If necessary, open the Database Cart. Choose **Window > Database > Database Cart**. For more information at any time, press F1 or click **Help** from within any of the windows or dialogs.
2. Click Open Cart on the Database Cart toolbar. For more information at any time, press F1 or click **Help** from within any of the windows or dialogs.
3. If there are objects already present in the Cart, a dialog asks whether you want to remove current objects from the cart before opening the saved cart.

   Yes empties the current cart and fills the cart with the objects from the cart you are opening.

   No does not empty the current cart, but adds the objects from the cart you are opening to the current cart contents. If one or more objects is a duplicate of the current cart contents it is not added (only one version of an identically named object can be present in a cart).
4. In the Open dialog, navigate to the saved XML file and click **Open**.

Examining Deployments in an Oracle Database Cloud Service Connection

You can examine deployments to an Oracle Database Cloud Service instance.

To examine a deployment:

1. If necessary, open the Databases window by choosing **Window > Database > Databases**.
2. Under the Cloud Connections node expand the Cloud connection node you want and then expand the Deployments node.
3. Double-click the deployment you want to examine. The Deployment Object Viewer window opens.
   - The Details tab shows information about the deployment.
   - The Log tab shows log information about the deployment.

To see information that is larger than the cell, hover the mouse over the cell. Alternatively, you can double-click the cell. The contents of the cell are displayed in a Details window. This is particularly useful for information like SQL*Loader files.

You can re-run a deployment, delete the log for a deployment, or delete a deployment from the Oracle Database Cloud Service instance.

To re-run a deployment:

- Right-click the deployment and choose Restart. The scripts contained in the original deployment are run again.

To clear the deployment logs:

- Right-click the deployment and choose Clear Logs. The information in the Log tab of the Deployment Object Viewer is deleted.

To delete a deployment:

- Right-click the deployment and choose Delete. The deployment is removed from the list of deployments.

> **Note:**
>
> This does not remove the database objects deployed to the Oracle Database Cloud Service instance. To do this, specify a deployment which uses a Before Script which contains the appropriate DROP statements for the objects you want to drop and deploy that.

Importing and Exporting Data

Learn how to import data into existing tables or new tables, import data using SQL*Loader, import data into external tables. Learn how to export data from databases using a database connection, and how to export data to files.

You can import data into tables in a database using a database connection.

> **Note:**
>
> You cannot import data 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:

- An 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.

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. See see *SQL*Loader Concepts*.

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. See *Using External Tables to Move Data*.

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 **Window > Database > Databases** to open the Databases window.
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.

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 **Window > Database > Databases** to open the Databases window.
2. If necessary, create a connection to the database.
3. Expand the node for the database connection and the schema. Right-click the Tables node and choose **Import Data**.
4. 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.

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 **Window > Database > Databases** to open the Databases window.
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.

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 and the schema. Right-click the Tables node and choose **Import Data** and in the Open dialog enter or browse to the location of the file.
3. Click **OK** to launch the Data Import Wizard.

   For more information at any time, press F1 or click Help **from** within the wizard.
4. On the Data Preview page of the Data Import wizard, choose External Table.
5. 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.

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. See *SQL*Loader Concepts*.
- `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 the TimesTen In-Memory Database page.

- `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.

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 **Window > Database > Databases** to open the Databases window.
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.

Copying, Comparing, and Exporting Databases

Using JDeveloper, you can copy database objects from a source schema to a destination schema, compare database schemas, and export database objects and data.

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:

- 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.

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 https://shop.oracle.com

You must have the source and the destination database connections already defined.

To compare database schemas:

- 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.

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:

- 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.

Working with Database Reports

View pre-defined reports about the database and its objects, and define new 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.

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 window 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.

How to Run Database Reports

The Database Reports window 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 window. In the main menu, choose **Window > 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.

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 window. In the main menu, choose **Window > 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.

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 window. In the main menu, choose **Window > 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.

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 window. In the main menu, choose **Window > 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.

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 window. In the main menu, choose **Window > 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

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 window. In the main menu, choose **Window > 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`.

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 window. In the main menu, choose **Window > 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.

To share 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 **Window > Database > Database Reports** to open the Database Reports window. The shared report is listed under the Shared Reports node in the Databases window.

Reference: Pre-Defined Database Reports

This section describes the pre-defined reports available under the Data Dictionary Reports node in the Database Reports window.

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 information, see *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.

Troubleshooting Database Connections

There a number of common problems that you may come across when working with databases. Learn what they are and see suggestions for solving them.

This section contains information to help you if you have problems connecting to a database.

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.

For information about the JDK, see "JDeveloper Certification Information" at `http://www.oracle.com/technetwork/developer-tools/jdev/documentation/index.html`.

For information about the JDK used by the database, consult your database documentation.

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/index.html`.

26

Designing Databases Within Oracle JDeveloper

This chapter describes how to work with database objects in a database connection. It also contains information about working 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:

- About Designing Databases Within Oracle JDeveloper
- Creating, Editing, and Dropping Database Objects
- Creating Scripts from Offline and Database Objects

About Designing Databases Within Oracle JDeveloper

Use JDeveloper to develop and work with databases.

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 figgeroisd 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. SeeModeling with Database Diagrams .

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 from a database connection.

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. Alternatively, you can have a connection to a live database and reverse engineer database objects. 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 (as part of a table)
- Database Link
- Materialized View
- Materialized View Log
- Package
- Procedure
- Sequence
- Synonym
- Table
- Tablespace
- Trigger
- Type
- View

For information about Oracle Database support of any of these object types, see *Database Objects* in *Introduction to Oracle SQL*

How to Work 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 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 reverse engineer 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 information about modeling databases, see Modeling with Database Diagrams .

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.

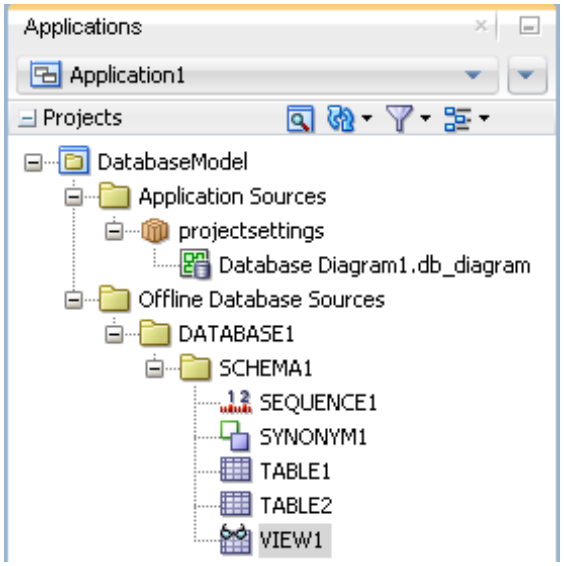
To set the database path for an existing project:

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.

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 Applications window, the offline database is shown below the Offline Database Sources node, shown in the following figure.

Figure 26-1 Offline Database in the Applications Window



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 of the offline database.

Configuring Offline Database Emulation

You can specify the type of database an offline database emulates. This determines the data types supported in the project.

For information about which database versions are compatible with JDeveloper, see the JDeveloper Certification Information at http://www.oracle.com/technetwork/developer-tools/jdev/documentation/index.html

How to Create Offline Databases

An offline database is a node in the Applications window that contains offline schemas and offline database object definitions.

To create an offline database:

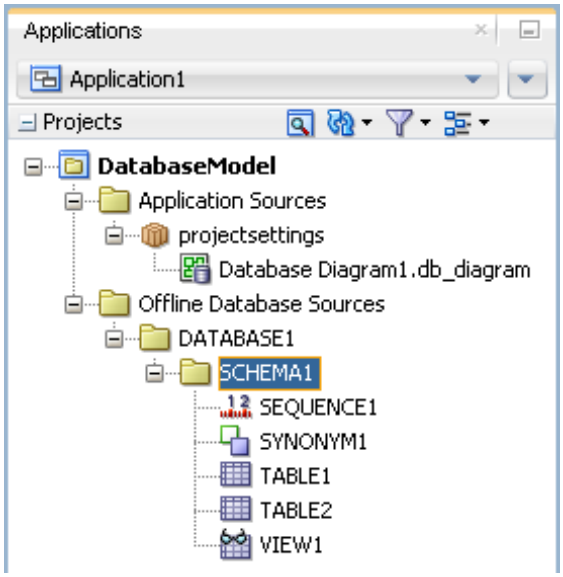
1. In the Applications window, locate the project you want to work in.
2. Right-click a project or anything in it, and choose **New** to display the New Gallery.
3. From the New Gallery, expand **Database Tier**, and select **Offline Database Objects**, and select **Offline Database**.
4. In the Create 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.

Offline Schemas

JDeveloper works with offline database definitions in the context of offline databases. Within the offline databases, offline schemas are the equivalent of schemas (or users) in live database connections. In the Applications window, the offline schema is shown below the Offline Database Sources node, illustrated in the following figure.

Figure 26-2 Offline Schema in Applications Window



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 `SCHEMA1`, which is in an offline database called `DATABASE1` under the Offline Database Sources node.

How to Create Offline Schemas

To create an offline schema:

1. In the Applications window, 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.

Example 26-1 Context Menu Shortcut:

In the Applications window, right-click the offline database, and choose **New Schema**.

How to Create Offline Database Objects

There are a number of ways that you can create offline database objects:

- You can always create offline database objects from the New Gallery.
- You can always create a database diagram, a table or a view from the context menu of a project configured for offline database development, or from a node under that project.
- Once you have created an offline database object, you can create another from the context menu of a project configured for offline database development, or from a node under that project.

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.

You can combine two partitioning methods, called composite partitioning, to further divide the data into subpartitions. Composite partitioning is supported for:

- Range-Range
- Range-Hash
- Range-List
- List-Range
- List-Hash
- List-List

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.

Index types are 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 or 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 BODY 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.

An 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 reverse engineering materialized views from Oracle Database to a JDeveloper project:

- If a materialized view on the database specifies WITHOUT REDUCED PRECISION, when it is reverse engineered 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 reverse engineered 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

To create an offline type definition:

1. In the Applications window, expand the application and project you want to work in.
2. Right-click a project or a node under it such as an offline schema, and choose **New > From Gallery** 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.

You can edit user-defined types by double-clicking the type in the Applications window. 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. In the Applications window, 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 Applications window, right-click the offline schema, select **New Database Object**, then select object you want to create.

To drop an offline database definition:

- In the Applications window, expand the project, offline database, and offline schema containing the offline object. 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.

How Reverse Engineer Database Definitions Based on Database Objects

You can drag tables, views, materialized views, synonyms, and sequences from an online database schema onto a database diagram, where they become accessible as offline database objects.

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:
   - **Window > Database > Databases window**.
   - **Application Resources** in the Applications window.

   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 Applications window.

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 reverse engineer 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 Reverse Engineer Database Objects and Offline Database Definitions to Projects

You can reverse engineer 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 reverse engineer 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 reverse engineering. 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 reverse engineer database objects:

1. In the Applications window, 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 **Offline Database Objects from Source Database** to launch the Offline Database Objects from Source Database wizard.

   For more information at any time, press F1 or click **Help** from within the wizard.

Offline Tables and Foreign Keys

When you reverse engineer 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.

Reverse Engineering Tables at Both Ends of a Foreign Key

This is the simplest case. When JDeveloper reverse engineers tables that have foreign keys between them, information about the foreign key is also reverse engineered. 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.

Best Practice

From the information above you can see that if you are reverse engineering 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 reverse engineering tables so that you can make the changes you want and then generate the changes back to the same database schema you should reverse engineer 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 Specify Operation page of the Offline Database Objects from Source Database wizard allows you to reverse engineer dependencies.

How to Refresh Offline Database Objects

You can refresh any reverse engineered offline object from the database connection it was originally reverse engineered from. Note that if you reverse engineer 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 reverse engineer the object from the database and overwrite the offline object by selecting the **Replace** on the Specify Operation page.

To reverse engineer an object from a database connection a second time:

1. Right-click the offline object in the Applications window, and choose **Refresh** from db-connection.
2. When the Confirm Offline Object Overwrite dialog appears, check that you want to reverse engineer the object and then click **Yes**. Otherwise click **No**. This may take a few seconds.

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 an existing offline database:

- In the Applications window, navigate to the template object and choose `Open` from the context menu. The template object opens in the appropriate editor where you an edit it.

How to Create Offline Database Objects from Templates

You can create offline database objects from templates as offline database objects in the Applications window, 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 Applications window, 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 Applications window under the offline database and schema.

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 user property libraries when creating offline database objects.

User property libraries can contain properties defined for:

- Tables, columns, constraints, indexes
- Database links
- Functions, packages, procedures
- Materialized Views
- Materialized View Logs
- Sequences
- Synonyms
- Tablespaces
- Triggers

- Types
- Views

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 Applications window, 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 values.

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. See Developing Extensions in Oracle JDeveloper and `UserPropertyValidationManager` in *Java API Reference for Oracle Extension SDK*.

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 the user property libraries.

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.

How to Generate Offline Database Objects to the Database

The Generate SQL from 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.

Alternatively, if you just want to generate one or more offline tables back to the database connection they were originally reverse engineered from, you can do this directly from the Applications window.

> **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.

Reconciliation issues

This section contains information about problems you may come across when reconciling.

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`.

Cannot reconcile renamed tables

You can change the name of a table when you reverse engineer 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 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 Applications window.
>
> 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 Applications window, 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 **Database Files**.
4. Select **SQL File from Source Database** to launch the Generate SQL from 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 Applications window, 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**.

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 reverse engineered 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.

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.

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.

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 Applications window under Resources.

> **Note:**
>
> If you specify a location that is outside the current project the reports are generated, but they are not listed in the Applications window. 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 Applications window, 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 Applications window under the offline database node.

To edit a predefined report:

1. In the Applications window, 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 Applications window, 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.

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 Applications window, 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 Applications window under the offline database node.

To run a report:

- In the Applications window, 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.

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:

```
<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.

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 Applications window, 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.

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. See UML-Offline Database Transformation

Working with Offline Database Objects in Source Control Systems

JDeveloper provides a number of features for developing in teams, including several version control software systems. These are described in Versioning Applications with Source Control.

Offline database 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 Database Objects wizard. For more information, see How to Generate Offline Database Objects to the Database .

Working with Database Objects

You can create database objects in a database connection in the Databases window.

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.

To create a database object in the Databases window:

1. If necessary, choose **Window > Database > Databases window**.
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 **Window > Database > Databases window**.
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 **Window > Database > Databases window**.

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**.

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 window. See Using the Database Reports Window.

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 reverse engineer tables from a database, date and time values are validated according to the rules above. If the validation prevents you from reverse engineering a table from Oracle Database, you can turn it off.

To turn off date and time validation:

1. Choose **Tools > Preferences > Database**.
2. Uncheck **Validate date and time default values**.

Creating Scripts from Offline and Database Objects

You can generate database objects and offline database definitions to SQL scripts, Oracle MetaBase (OMB) files which can be imported into Oracle Warehouse Builder, or SXML files.

How to Create SQL Scripts

You can create SQL scripts from offline database definitions or from database objects.

The script is generated with the default name *script1.sql*. It is opened in the SQL Worksheet, and listed in the Applications window under the Resources node for the current project.

To create a SQL script from the Databases window or Applications Window:

1. Choose **Window > Database > Databases window**, expand the database connection and schema, and right-click the database object you want to create the script from.

   or

   Choose **Window > Applications**, navigate to the offline database definition you want to create the script from, and right-click the database object you want to create the script from.
2. Choose **Generate to > SQL script**.
3. The Generate SQL from Database Objects wizard opens where you specify the details of how to create the script. For more information at any time, press F1 or click **Help** in the dialog.
4. When you click **Finish**, the script is created and opened in the source editor.

To create a SQL script from the New Gallery:

1. Open the New Gallery by choosing **File > New**.
2. In the New Gallery, in the **Categories** tree, under **Database Tier**, select **Database Files**.
3. In the **Items** list, double-click **SQL Script from Source Database**.
4. In the Generate SQL Script from Database Objects wizard, enter details of the script, the source and the objects.

   For help with the wizard, press F1 or click **Help**.
5. When you click **Finish**, the script is created and opened in the source editor.

How to Create OMB Scripts from Tables

You can create files formatted as Oracle MetaBase (OMB) scripts for Oracle Warehouse Builder from offline tables in the Applications window.

The file is generated with the default name *omb_scriptn.tcl*. It is opened in the source editor, and listed in the Applications window under the Resources node for the current project.

To create a OMB script from the Applications window:

1. Choose **Window > Applications**, navigate to the offline table or tables you want to create the file from.
2. Right-click and choose **Generate to > OMB script**.

   The file is created and opened in the source editor.

To create a OMB script from the New Gallery:

1. Open the New Gallery by choosing **File > New**.
2. In the New Gallery, in the **Categories** tree, under **Database Tier**, select **Database Files**.
3. In the **Items** list, double-click **OMB File from Source Database**.
4. In the Generate OMB Script from Database Objects wizard, enter a name for the file and select the source offline database and click **Next**.

   For help with the wizard, press F1 or click **Help**.

5. On the Select Objects page, choose the offline objects to include in the file, then click **Finish**.

   The file is created and opened in the source editor.

How to Create SXML Scripts

You can create SXML files from offline tables in the Applications window.

A script is generated for each offline database object with the name *object-name_object-type.xml*. The scripts are opened in the XML Source Editor, and listed in the Applications window under the offline database node.

To create SXML files from the Applications window:

1. Choose **Window > Applications**, navigate to the offline table or tables you want to create the script from.
2. Right-click and choose **Generate to > SXML**.

   The files are created and opened in the source editor.

To create SXML files from the New Gallery:

1. Open the New Gallery by choosing **File > New**.
2. In the New Gallery, in the **Categories** tree, under **Database Tier**, select **Database Files**.
3. In the **Items** list, double-click **SXML File from Source Database**.
4. In the Generate SXML File from Database Objects wizard, enter a name for the script and select the source offline database and click **Next**.

   For help with the wizard, press F1 or click **Help**.
5. On the Select Objects page, choose the offline objects to include in the file, then click **Finish**.

   The file is created and opened in the source editor.

27

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:

- About Using Java in the Database
- Choosing SQLJ or JDBC
- Using Java Stored Procedures

About Using Java in the Database

Use Java with a database. You can embed SQL in Java programs using SQLJ or JDBC. You can access database objects and packages from Java programs. Use Java stored procedures, which are Java methods 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.

Choosing SQLJ or JDBC

Choose whether to use SQLJ or JDBC to embed SQL in Java programs.

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.

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. See *Oracle® Database SQLJ Developer's Guide*.

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.

Using Oracle JDBC Drivers

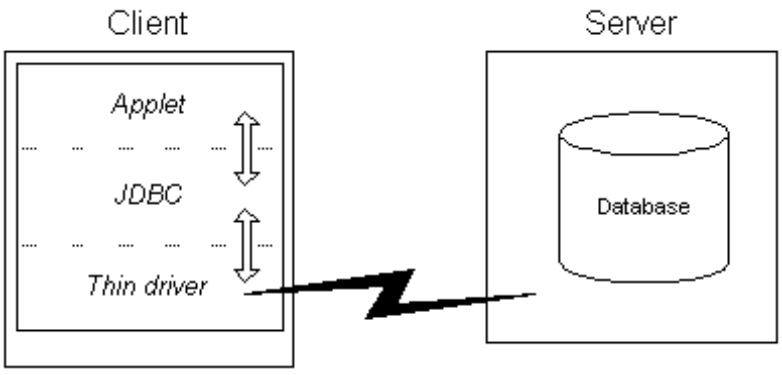
JDBC provides Java programs with low-level access to databases.

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

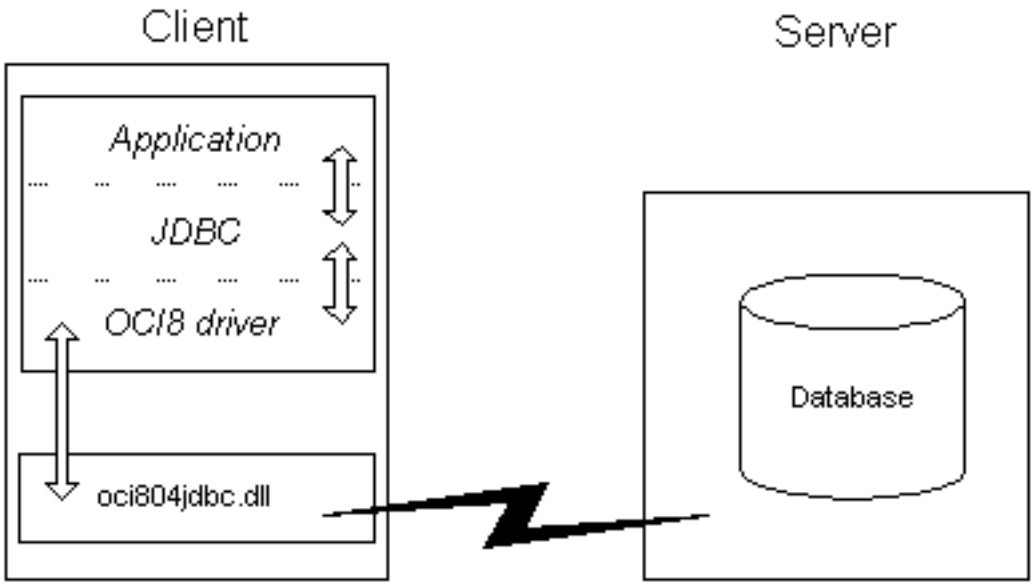
  The following figure illustrates how JDBC components and the driver run in the same memory space as an applet.

Figure 27-1 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 27-2 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.

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.

Embedding SQL in Java Programs with SQLJ

You have to perform a number of tasks to embed SQL in Java programs with SQLJ.

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 Applications window, 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.

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 Applications window, 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.

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 in the Compiler > SQLJ page of the Project Properties dialog.
2. In the Applications window, 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.

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.

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:

- 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.

How to Create a Connection Context Object

Before it can be used in an SQLJ statement, create a declared connection.

Named connection contexts are not required: SQLJ statements that omit the connection context name use the default connection context.

To create a context object, use a statement to create an instance (for example, `thisCtx`) for the connection context class (for example, `MyConnectionContext`):

```
MyConnectionContext thisCtx = new MyConnectionContext (<myPath>, <myUID>,
<myPasswd>, autocommit)
```

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. See *SQLJ Debugging Features*.

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 Applications window, 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. These include:
   - The level at which translater warnings should be set.
   - Type of code generation.
   - Whether you want to perform SQL semantic checking against a database schema.
   - Additional options to be used in the SQLJ translator.
4. Click **OK**.

You can set SQLJ translator properties for all projects by choosing **Default Project Properties** from the Application menu

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.

Embedding SQL in Java Programs with JDBC

JDBC provides Java programs with low-level access to databases. See *Oracle® Database SQLJ Developer's Guide*.

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 (Transparent Application Failover) or the enhanced proxy features like middle-tier authentication, then choose the OCI driver.

How to Modify a Project to Use a Non-Default JDBC Driver

If your JDeveloper programming environment has 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 Applications window, 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.

How to Code a JDBC Connection

You can establish a database connection in pure JDBC code.

See Oracle® Database SQLJ Developer's Guide for a summary.

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...);
```

Using Java Stored Procedures

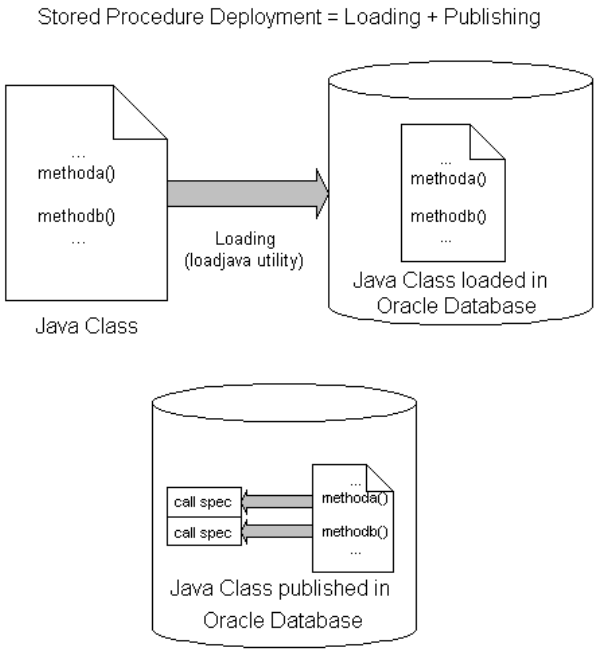
Java stored procedures are Java methods that reside and run in a database. Stored procedures can help improve the performance of database applications because they are efficient: they are stored in the RDBMS in executable form, and run in the RDBMS (rather than the client) memory space.

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. See *Developing Java Stored Procedures*.

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 27-3 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 Applications window, 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 `.dbexport` 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.dbexport` deployment profile appears in the Applications window below the specified project.
6. Select and right-click `storedProc.dbexport` in the Applications window. 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.
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 Applications window. 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 Applications window. 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 Applications window below the specified project.
2. From the context menu, choose `Export 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 Applications window 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 Hello() from dual;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!
```

How to Debug Java Stored Procedures

Debug Java stored procedures through a database connection.

To debug PL/SQL:

1. Choose **Window > Database > Databases window**.
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.

How to Remove Java Stored Procedures

To drop a stored procedure:

1. Choose **Window > Database > Databases window**.
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**.

28

Running and Debugging PL/SQL and Java Stored Procedures

This chapter describes how to run and debug PL/SQL and Java stored procedures. This chapter includes the following sections:

- About Running and Debugging PL/SQL and Java Stored Procedures
- Running and Debugging Functions_ Procedures_ and Packages
- Debugging PL/SQL Programs and Java Stored Procedures

About Running and Debugging PL/SQL and Java Stored Procedures

Learn how to run and debug Java in a database.

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.

Running and Debugging Functions, Procedures, and Packages

Learn about 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 **Window > Database > Databases window**.
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 Databases window is currently selected.
5. Click **Debug**, or if you want to run without debugging, click **Run**.
6. The Run PL/SQL dialog is displayed.
   a. 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

   b. The **Parameters** box lists the target's arguments (if applicable).
   c. 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.

Debugging PL/SQL Programs and Java Stored Procedures

Learn how to debug PL/SQL programs and Java stored procedures in Oracle Databases.

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.

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. See Debugging PL/SQL and Java Stored Procedures Prerequisites.

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.

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.

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.

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.

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 Applications window 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;`

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.

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 **Window >Database > Databases window**.
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 Applications window.
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 Debugging Java Programs.
10. Resume debugging the PL/SQL program until you are satisfied.

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 remote debugger with PL/SQL programs and Java stored procedures in Oracle Database.

Make sure that you've completed the documented prerequisites, listed in 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 Applications window, 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 Databases window, right click the connection and chose **Remote Debug**.
9. In the Databases window, 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 Applications window, 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 Processes 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 Processes window. 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 Processes window, 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 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 Debugging Java Programs.
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 Processes window and choose **Stop Listening**.

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 28-1 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`