

Oracle® Enterprise Pack for Eclipse

User's Guide

12c (12.1.3.4)

E58434-01

March 2015

Describes how to use Oracle Enterprise Pack for Eclipse, which is a set of plugins for Eclipse, to support Java EE development. It allows you to create, configure and deploy Oracle Mobile Application Framework applications for iOS and Android. You can create, configure, and run Oracle ADF applications on Glassfish and Oracle WebLogic Server.

Oracle Enterprise Pack for Eclipse User's Guide, 12c (12.1.3.4)

E58434-01

Copyright © 2008, 2015, Oracle and/or its affiliates. All rights reserved.

Primary Author: Catherine Pickersgill

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 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 installed on the hardware, and/or documentation, delivered to U.S. Government end users are "commercial computer software" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. 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 and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Xeon 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, Opteron, the AMD logo, and the AMD Opteron 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	xi
Audience	xi
Documentation Accessibility	xi
Related Documents	xi
Conventions	xi
1 Oracle Enterprise Pack for Eclipse User's Guide	
1.1 Oracle Application Development Framework Support	1-1
1.2 Oracle Mobile Application Framework Support	1-2
1.3 Oracle WebLogic Server Support	1-2
1.4 Integrating Oracle Cloud Services	1-2
1.5 Maven Support	1-2
1.6 Web Services Support	1-3
1.7 REST Web Services Support	1-3
1.8 Oracle Database Support	1-3
1.9 Object-Relational Mappings Support	1-3
1.10 Spring Support	1-3
1.11 Coherence Support	1-4
1.12 Web Application Development Support	1-4
2 Oracle ADF Tools Support	
2.1 Getting Started with Oracle ADF	2-1
2.1.1 Understanding the Oracle ADF Architecture	2-2
2.1.2 Configuring Oracle WebLogic Server	2-2
2.1.3 Creating an Oracle ADF Application	2-4
2.2 Working with the Oracle ADF Model Layer	2-5
2.2.1 Creating the JPA Model Project	2-6
2.2.2 Creating JPA Entities from Tables	2-6
2.2.3 Working with Session Beans	2-7
2.2.3.1 Generating a Session Bean on Selected JPA Entities	2-7
2.2.3.2 Generating a JSF Managed Bean	2-7
2.2.3.3 Generating a Session Bean and a JSF Managed Bean using the Data Components Model Wizard 2-8	
2.2.3.4 Editing a Session Bean	2-8
2.2.4 Working with ADF Model Data Binding	2-8

2.2.4.1	Creating ADF Data Controls.....	2-10
2.2.4.2	Using the ADF Data Control Manager.....	2-12
2.2.4.3	Using the Data Palette to Create UI Components	2-15
2.2.4.4	Using the Bindings Tab in the Properties Window	2-17
2.2.4.5	Working with Page Definition Files.....	2-17
2.2.4.6	Opening a Page Definition File in the Page Definition Editor	2-18
2.2.4.7	Understanding the Page Definition Editor	2-18
2.2.4.8	Understanding Bindings and Executables.....	2-20
2.2.4.9	Adding Bindings and Executables.....	2-22
2.2.4.10	Working with Tree Bindings.....	2-22
2.2.5	Adding Data Binding to Existing UI Components	2-24
2.2.6	Debugging ADF Bindings	2-26
2.2.6.1	ADF Page Definition Artifact Validation.....	2-26
2.2.7	Refactoring ADF Bindings.....	2-27
2.3	Working with Oracle ADF Controller	2-27
2.3.1	Understanding ADF Task Flows	2-27
2.3.2	Creating a New Task Flow	2-27
2.3.3	Adding Activities to a Task Flow	2-28
2.3.4	Adding ADF Bindings to a Task Flow.....	2-28
2.3.5	Adding Control Flows to a Task Flow.....	2-32
2.3.6	Using Task Flows as Regions.....	2-32
2.3.7	Running an ADF Task Flow.....	2-33
2.4	Working with Oracle ADF Faces	2-33
2.4.1	About ADF Faces Configuration Files.....	2-34
2.4.2	About ADF Data Visualization Components	2-34
2.4.3	Working with ADF tags in JSP Pages	2-34
2.4.4	Support for ADF Components in the Palette.....	2-34
2.4.5	Using the Tag Drop Editor for ADF Faces Components	2-35
2.4.6	Using the Smart Editor for ADF Components	2-35
2.5	Deploying an Oracle ADF Application	2-35
2.6	Debugging an Oracle ADF Application	2-36
2.6.1	Using ADF Source Code with the Debugger.....	2-36
2.6.2	Setting ADF Declarative Breakpoints	2-37
2.6.3	Setting and Using ADF Task Flow Breakpoints.....	2-37
2.6.4	Setting and Using ADF Page Definition Breakpoints	2-38
2.6.5	Setting and Using ADF Lifecycle Phase Breakpoints.....	2-39
2.6.6	Using the EL Expression Evaluator	2-40
2.6.7	Using the ADF Structure and Data Window	2-41
2.6.7.1	Using the ADF Structure Pane.....	2-41
2.6.7.2	Using the ADF Data Pane.....	2-41
2.7	Using AppXray for Oracle ADF Artifacts	2-42
2.8	Refactoring Oracle ADF Components	2-42
2.8.1	Refactoring ADF Pages	2-42
2.8.2	Refactoring ADF Task Flow configuration files.....	2-43
2.8.3	Refactoring JSF/ADF Managed Beans	2-44
2.8.4	Refactoring ADF Data Binding Artifacts.....	2-44
2.8.5	Externalizing Strings	2-45

2.8.6	Adding and Refactoring ADF Tag IDs	2-46
2.9	Reusing Oracle ADF Application Components	2-46
2.9.1	About ADF Library Support	2-46
2.9.1.1	Naming Conventions	2-46
2.9.2	Creating an ADF Library	2-47
2.10	Configuring and Using ADF with GlassFish Server	2-47
2.10.1	How to Download ADF Essentials	2-47
2.10.2	How to Download and Install GlassFish Server	2-48
2.10.3	How to Configure GlassFish for OEPE	2-49
2.10.4	How to Configure GlassFish for ADF Essentials	2-49
2.10.4.1	Installing ADF Essentials on a Domain.....	2-49
2.10.4.2	Installing ADF Essentials on a Domain With a Password.....	2-50
2.10.5	How to Register the ADF Essentials Client WAR Library in Your Workspace	2-51
2.10.6	How to Create an ADF Application that Uses GlassFish Runtime	2-52
2.10.7	How to Create a Global JDBC Data Source	2-55
2.10.8	Known Problems and Solutions	2-56
2.11	Appendix A Oracle ADF XML Files	2-57
2.11.1	Oracle ADF Data Binding Files.....	2-57
2.11.2	Web Configuration Files	2-58

3 Oracle MAF Tools Support

3.1	Developing with Oracle MAF	3-1
-----	----------------------------------	-----

4 Oracle WebLogic Server Support

4.1	Feature Overview.....	4-1
4.2	WebLogic Shared Libraries	4-2
4.2.1	Common Operations.....	4-2
4.2.1.1	Adding a New Library to the Registry.....	4-2
4.2.1.2	Adding a Library Reference to the Project Classpath	4-3
4.2.1.3	Modifying a Library Reference on the Project Classpath.....	4-3
4.2.1.4	Removing a Library Reference from the Project Classpath.....	4-3
4.2.2	Validation Problems	4-4
4.2.2.1	Validation Errors	4-4
4.2.2.2	Validation Warnings	4-6
4.3	Support for WebLogic SCA	4-6
4.3.1	Configuring Projects to Use WebLogic SCA	4-6
4.3.2	Using Context Help for WebLogic SCA XML Attributes	4-7
4.3.3	Creating Complex Properties Using XML Template.....	4-8
4.3.4	Creating WebLogic SCA Data-Binding Customization Descriptor	4-8
4.3.5	Deploying a WebLogic SCA Application.....	4-9
4.3.6	Running a WebLogic SCA Application.....	4-9
4.4	Support for WebLogic Scripting Tool (WLST)	4-9
4.4.1	Configuring Projects for WLST.....	4-9
4.4.2	Creating New WLST Files	4-10
4.4.3	Editing WLST Script.....	4-10
4.4.4	Adding WLST Templates	4-10

4.4.5	Navigating MBean Structures.....	4-11
4.4.6	Using WLST Console	4-11
4.4.7	Executing WLST.....	4-11
4.4.8	Debugging WLST Script	4-12
4.4.9	Importing Existing WLST Script into OEPE	4-12
4.4.10	Known Issues and Limitations	4-12
4.5	Editing Deployment Descriptors	4-12
4.5.1	Using Deployment Descriptor Editors	4-12
4.5.1.1	Editor Keyboard Navigation.....	4-13
4.5.2	Creating JMS Descriptors	4-14
4.6	Using Deployment Plan Editor	4-14
4.6.1	Creating a New Deployment Plan	4-14
4.6.2	Editing a Deployment Plan	4-15
4.6.3	Using an Existing Deployment Plan to Configure an Application	4-16

5 Integrating Oracle Cloud Services

5.1	Adding Your Oracle Cloud Services.....	5-1
5.1.1	Using the Cloud View	5-2
5.2	Getting up and Running with Your Java Cloud Service.....	5-3
5.2.1	Viewing the Java Cloud Service Jobs Log	5-4
5.2.2	Viewing the Java Cloud Service Instance Log.....	5-5
5.3	Validating with the Whitelist Scan.....	5-6
5.4	Deploying to the Cloud.....	5-6
5.5	Oracle Developer Cloud Service.....	5-6
5.5.1	Logging In to Oracle Developer Cloud Service.....	5-6
5.5.2	Getting Up and Running with Your Developer Cloud Service	5-7
5.5.3	Using the Oracle Developer Services Cloud View	5-7
5.5.4	Importing an Oracle Developer Cloud Service Project.....	5-8
5.5.5	Exporting a Project from OEPE to Oracle Developer Cloud Service	5-8
5.5.6	Using eGit for DCS Source Control and Versioning	5-8
5.5.7	Using Git Tools in OEPE.....	5-9
5.5.8	Committing Changes to Oracle Developer Cloud Service Git Repository	5-9
5.5.9	Pushing Changes From the Local Git Repository to Oracle Developer Cloud Service Git Repository	5-10
5.5.10	Managing Documentation.....	5-10
5.5.11	Updating Tasks	5-10
5.5.11.1	Importing Tasks from Oracle Developer Cloud Service With a Custom Query	5-10
5.5.11.2	Creating a Local Task	5-11
5.5.11.3	Editing a Task.....	5-11
5.5.11.4	Synchronizing Tasks with Oracle Developer Cloud Service	5-11
5.5.11.5	Associating a Task with a Commit Transaction.....	5-11
5.5.12	Monitoring Hudson Builds	5-11

6 Maven Support

6.1	Using Maven with OEPE	6-1
6.2	Setting up Your Maven Environment.....	6-2

6.2.1	How to Set Up Your Maven Environment.....	6-2
6.3	Creating a Maven Settings File	6-3
6.3.1	How to Create Your Maven Settings File.....	6-3
6.4	Populating the Maven Repository.....	6-4
6.4.1	How to Use the Oracle Maven Synchronization Plug-In	6-4
6.4.2	Running the Oracle Maven Synchronization Plug-in	6-5
6.4.2.1	Populating a Local Repository.....	6-6
6.4.2.2	Populating a Remote Repository.....	6-7
6.4.2.3	What Happens When You Run a Push Goal to Populate a Repository?	6-8
6.5	Installing the Maven Archetypes.....	6-8
6.5.1	How to Install the Maven Archetypes.....	6-9
6.6	Creating ADF Applications with Maven Integration.....	6-9
6.6.1	How to Create an ADF Application with Maven Integration from the Command Line..	6-10
6.6.2	How to Create an Maven Project with Maven integration from the Wizard	6-10
6.6.3	How to Add Maven Integration to New ADF Application Projects.....	6-12
6.7	Importing Maven Projects	6-13
6.8	Using Maven to Deploy to a WebLogic Server	6-15
6.8.1	How to Deploy using Maven to a Running WebLogic Server.....	6-15

7 Web Services Support

7.1	Starting Points of Web Services Development with OEPE.....	7-1
7.1.1	Generating a Web Service From a WSDL File	7-1
7.1.1.1	Customizing a Web Service	7-2
7.1.2	Generating a Web Service From Java	7-6
7.1.2.1	Creating a Web Service from a Java Class	7-6
7.1.2.2	Creating a Web Service From Scratch Using Java	7-7
7.1.3	Generating a WSDL File	7-7
7.1.4	Contents of a WSDL File.....	7-8
7.1.5	Imported WSDL Files.....	7-8
7.1.6	Creating a New WSDL File	7-8
7.1.7	Understanding Policy Stores.....	7-9
7.1.8	Testing Web Services.....	7-9
7.2	Creating Web Services Projects.....	7-10
7.2.1	Creating a new Web Service Project	7-10
7.2.2	Creating a Web Service Project From an Existing Dynamic Web Project	7-11
7.3	Generating Client Code for Web Services	7-11
7.3.1	Generating Client Code From a WSDL File.....	7-11
7.3.2	Generating Client Code from a Java Class.....	7-12
7.3.3	Alternative Ways to Generate the Client Code	7-13
7.3.4	Deploying Java Web Service Applications to Oracle WebLogic Server	7-13
7.4	Generating JAXB Types	7-14
7.5	Using Client Proxy Templates	7-14
7.6	Using WebLogic Web Services Annotations View	7-16
7.6.1	Activating the WebLogic Web Services Annotations View	7-16
7.6.2	Using the WebLogic Web Services Annotations View	7-16
7.6.3	Supported Annotations.....	7-17

7.7	Validating Web Services Projects	7-18
7.7.1	Validated Resources	7-18
7.7.2	Configuring Project Validation.....	7-19
7.8	Generating Web Services for Spring Service Beans	7-19
7.9	Configuring HTTPS Client Credentials.....	7-20

8 REST Web Services Support

8.1	Getting Started with REST Web Services	8-1
8.2	Creating Projects Configured for REST	8-2
8.2.1	How to Create a Dynamic Web Project that is Configured for REST	8-2
8.2.2	How to Configure a Java Project for REST	8-4
8.3	Creating a REST Web Service	8-5
8.3.1	How to Create a Patterned REST Web Service.....	8-6
8.3.2	How to Create a POJO REST Web Service.....	8-8
8.4	Mapping Incoming Requests to Java Methods.....	8-10
8.4.1	How to Map an HTTP Request to Java Methods in the REST Generation Wizard	8-10
8.4.2	How to Map HTTP Requests to Java Methods in the Java Class	8-10
8.4.3	How to Map HTTP Requests to Java Methods in the Annotations View	8-11
8.5	Customizing Media Types for the Request and Response Messages	8-11
8.5.1	How to Customize Media Types in the Java Source Editor	8-12
8.5.2	How to Customize Media Types in the Annotations View for a Java Class.....	8-12
8.6	Validation and Quick Fix.....	8-13
8.7	Content Assist.....	8-14
8.8	Run-AS JAX-RS Support.....	8-14
8.8.1	How to Deploy to a Targeted Runtime J2EE Server.....	8-15
8.8.2	How to Deploy to a Basic HTTP Lightweight Server.....	8-15
8.9	Generate a Java REST Client from a WADL	8-17

9 Oracle Database Support

9.1	Getting Started with the Oracle Database Plugin for Eclipse.....	9-1
9.1.1	Using the Database Explorer.....	9-1
9.1.1.1	Creating a Connection to a Database.....	9-1
9.1.1.2	Working with a Database Connection.....	9-2
9.1.1.3	Editing Data in a Table	9-2
9.1.1.4	Loading Data into a Table	9-3
9.1.1.5	Extracting Data from a Table	9-3
9.1.1.6	Generating DDL.....	9-3
9.1.2	SQL Tools	9-3
9.1.2.1	Using SQL Editor.....	9-4
9.1.2.2	Executing a Stored Procedure or Function	9-4
9.1.2.3	Executing Explain Plans	9-4
9.1.3	Granting and Revoking Privileges	9-4
9.1.4	Creating Tables	9-5
9.1.5	Troubleshooting	9-6
9.2	Using the RDB Schema Editor	9-7
9.2.1	How to Display a Database Schema in the Editor	9-7
9.2.2	Working with RDB Schema Editor Features	9-8

10	Object-Relational Mappings Support	
10.1	Configuring a JPA Project to Use EclipseLink Persistence Provider.....	10-1
10.2	Configuring a JPA Project to Use Kodo Persistence Provider.....	10-3
10.3	Oracle WebLogic Server Support for Persistence Provider Libraries and Deployment	10-4
11	Spring Support	
11.1	Generating Spring Artifacts.....	11-1
11.2	Generating Web Services for Spring Service Beans	11-4
12	Coherence Support	
12.1	Coherence Tooling: Configuring Projects for Coherence.....	12-1
12.1.1	Configuring Coherence Facet	12-1
12.1.2	Editing Coherence Launch Configuration	12-2
12.1.3	Editing Coherence Operational Configuration	12-3
12.1.4	Editing Coherence Cache Configuration.....	12-3
12.2	Working with Coherence (GAR) Applications.....	12-3
12.2.1	Creating Coherence Applications	12-4
12.2.2	Exporting a Coherence Application.....	12-4
12.2.3	Deploying a Coherence Application.....	12-5
12.2.4	Locating Your Deployed Coherence Application.....	12-5
13	Web Application Development Support	
13.1	Using AppXray Technology	13-1
13.1.1	Enabling and Disabling AppXray	13-2
13.1.2	Visualizing AppXray Dependencies.....	13-2
13.2	Configuring JSF Projects	13-3
13.2.1	Supported JSF Libraries and Versions.....	13-5
13.2.2	Creating a Faces Configuration File.....	13-5
13.2.3	Using the Faces Configuration Node.....	13-5
13.2.3.1	Creating a New Managed Bean.....	13-6
13.2.3.2	Creating a New Navigation Case.....	13-6
13.2.3.3	Creating a New Converter	13-7
13.2.3.4	Creating a New Validator	13-7
13.2.4	Using the Faces Configuration Editor	13-7
13.2.5	Understanding JSF Resource Bundles	13-8
13.3	Configuring JSTL Projects.....	13-8
13.3.1	Supported JSTL Libraries and Versions	13-9
13.4	Configuring Projects for Apache Trinidad.....	13-9
13.4.1	Trinidad Library Support by the Trinidad Facet	13-10
13.5	Configuring Projects with External Resources	13-11
13.5.1	Using a Dynamic Project.....	13-11
13.5.2	Using Linked Resources	13-11
13.5.3	Configuring a Deployment Assembly	13-11
13.6	Creating a JSF Project From an Existing Web Project	13-12
13.7	Using the Web Page Editor.....	13-12

13.7.1	Using the Design View	13-13
13.7.2	Using the Preview Tab	13-13
13.7.3	Using the Source View	13-14
13.7.3.1	Using the Content Assist	13-14
13.7.3.2	Using HyperLink	13-14
13.7.3.3	Using HoverHelp	13-14
13.7.4	Using the Outline View	13-14
13.8	Editing Tags Using Property Sheets.....	13-14
13.8.1	Choosing Binding	13-15
13.8.2	Choosing a Method	13-15
13.8.3	Selecting a Navigation Case	13-15
13.8.4	Selecting a File	13-15
13.8.5	Selecting a Style Class	13-15
13.8.6	Defining CSS Style	13-15
13.8.7	Choosing a Resource Bundle	13-16
13.8.8	Choosing a Validator.....	13-16
13.8.9	Choosing a Converter	13-16
13.9	Using the Web Page Editor Palette.....	13-16
13.9.1	Displaying the Palette in External View.....	13-17
13.9.2	Editing Tag Library Entries in the Palette.....	13-17
13.9.3	Using the Data Palette.....	13-17
13.9.4	Customizing the Palette.....	13-18
13.9.5	Docking and Undocking the Palette	13-18
13.9.6	Modifying the Display of the Palette	13-18
13.10	Enabling Localization in the Web Page Editor	13-19
13.11	Creating JSF HTML Tags	13-19
13.11.1	Adding a PanelGrid Tag.....	13-20
13.11.2	Adding a dataTable Tag	13-20
13.11.3	Adding a form Tag	13-21
13.12	Generating Struts Artifacts	13-22
13.12.1	Configuring a Project for Struts	13-22
13.12.2	Generating Struts Files and Updating the Configuration.....	13-23
13.13	Supported Versions	13-23

Preface

Welcome to the *Oracle Enterprise Pack for Eclipse User's Guide*.

Audience

This document is intended for application developers who develop applications using Oracle Enterprise Pack for Eclipse.

Documentation Accessibility

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

Access to Oracle Support

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

Related Documents

For more information, see the following documents:

- *Installing Oracle Enterprise Pack for Eclipse*
- *Oracle Enterprise Pack for Eclipse Online Help*
- *Developing Mobile Applications with Oracle Mobile Application Framework (OEPE Edition)*

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.
<i>italic</i>	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.

Convention	Meaning
monospace	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

Oracle Enterprise Pack for Eclipse User's Guide

Oracle Enterprise Pack for Eclipse (OEPE) is a set of plug-ins designed for the Eclipse IDE to support Java EE development.

This document describes the following key features of OEPE:

- [Oracle Application Development Framework Support](#)
- [Oracle Mobile Application Framework Support](#)
- [Oracle WebLogic Server Support](#)
- [Integrating Oracle Cloud Services](#)
- [Maven Support](#)
- [Web Services Support](#)
- [REST Web Services Support](#)
- [Oracle Database Support](#)
- [Object-Relational Mappings Support](#)
- [Spring Support](#)
- [Coherence Support](#)
- [Web Application Development Support](#)

1.1 Oracle Application Development Framework Support

OEPE provides a set of plugins for the Eclipse IDE designed to create, configure, and run Oracle ADF applications.

The Oracle Application Development Framework (Oracle ADF) is an end-to-end application framework that builds on Java Platform, Enterprise Edition (Java EE) standards and open-source technologies to simplify and accelerate implementing service-oriented applications. If you develop enterprise solutions that search, display, create, modify, and validate data using web, wireless, desktop, or web services interfaces, Oracle ADF can simplify your job. Used in tandem, OEPE and Oracle ADF give you an environment that covers the full development lifecycle from design to deployment, with drag and drop data binding and visual UI design features built in.

For information on ADF runtimes see, [Configuring Oracle WebLogic Server](#) and [Configuring and Using ADF with GlassFish Server](#).

For more information, see [Oracle ADF Tools Support](#).

1.2 Oracle Mobile Application Framework Support

Oracle Mobile Application Framework (MAF) allows you to create mobile applications that run natively on both iOS and Android phones and tablets.

Oracle MAF is a hybrid mobile architecture, one that uses HTML5 and CSS to render the user interface in the web view, Java for the application business logic, and Apache Cordova to access device features such as GPS activities and e-mail.

For more information, see [Oracle MAF Tools Support](#).

1.3 Oracle WebLogic Server Support

Oracle WebLogic Server is a scalable, enterprise-ready Java Platform, Enterprise Edition (Java EE) application server. The WebLogic Server infrastructure supports the deployment of many types of distributed applications and is an ideal foundation for building applications.

The WebLogic Server complete implementation of the Java EE 5.0 specification provides a standard set of APIs for creating distributed Java applications that can access a wide variety of services, such as databases, messaging services, and connections to external enterprise systems. End-user clients access these applications using Web browser clients or Java clients. It also supports the Spring Framework, a programming model for Java applications which provides an alternative to aspects of the Java EE model.

OEPE enables you to develop, deploy, and debug Oracle WebLogic Server applications. You can:

- Develop applications faster with virtual EAR technology.
- Deploy applications remotely.
- Use the OEPE support for WebLogic Shared Libraries.
- Use the OEPE support for WebLogic SCA.
- Use the OEPE support for WLST.
- Use the OEPE support for XMLBeans.
- Use EJBGen.

For more information, see [Oracle WebLogic Server Support](#).

1.4 Integrating Oracle Cloud Services

Oracle Cloud offers a broad portfolio of software as a service applications, platform as a service, and social capabilities, all on a subscription basis. Oracle Cloud delivers instant value and productivity for end users, administrators, and developers alike through functionally rich, integrated, secure, enterprise cloud services.

You can develop applications in OEPE and deploy them to Oracle Cloud. For more information, see [Integrating Oracle Cloud Services](#)

1.5 Maven Support

Maven is an open source build management tool that is central to project build tasks such as compilation, packaging, and artifact management. Maven uses a strict XML-based rule set to promote consistency while maintaining flexibility. Most Java-centric continuous integration systems integrate well with Maven, making it a

good choice for an underlying build system. This chapter describes how use Maven for Oracle Enterprise Pack for Eclipse (OEPE).

For more information, see [Maven Support](#).

1.6 Web Services Support

Web services are programs that can be accessed remotely using different XML-based languages. What these programs can do (that is, the functionality they implement) is described in a standard XML vocabulary called Web Services Description Language (WSDL). For example, a banking Web service may implement functions to check an account, print a statement, and deposit and withdraw funds. These functions are described in a WSDL file that any consumer can invoke to access the banking Web service. As a result, a consumer does not have to know anything more about a Web service than the WSDL file that describes what it can do.

OEPE lets you to build enterprise-class Web services that employ standard Web service technologies, such as XML, SOAP, and WSDL. OEPE simplifies Web service development by allowing you to focus on application logic, rather than the complex implementation details traditionally required by these technologies.

For more information, see [Web Services Support](#).

1.7 REST Web Services Support

You can create and run web services in OEPE which conform to the Representational State Transfer (REST) architectural style using Java API for RESTful Web Services (JAX-RS)

For more information, see [REST Web Services Support](#).

1.8 Oracle Database Support

The OEPE database support lets you easily connect to, create, explore, and query Oracle databases. Support includes database visualization through the Data Source Explorer view and DDL generation.

For more information, see [Oracle Database Support](#).

1.9 Object-Relational Mappings Support

JPA, part of the Java EE 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. JPA works both inside a Java EE application server and outside an EJB container in a Java Standard Edition (Java SE) application.

With OEPE, you can create a persistence layer that uses EJB 3.0 JPA.

For more information, see [Object-Relational Mappings Support](#).

1.10 Spring Support

Using OEPE, you can generate Spring configuration and beans from persistence mappings.

For more information, see [Spring Support](#).

1.11 Coherence Support

OEPE provides support for Oracle Coherence. Coherence provides replicated and distributed (partitioned) data management and caching services on top of a peer-to-peer clustering protocol.

For more information, see [Coherence Support](#).

1.12 Web Application Development Support

OEPE simplifies various aspects of your Web application development process.

For more information, see [Web Application Development Support](#).

Oracle ADF Tools Support

OEPE provides a set of plugins for the Eclipse IDE designed to create, configure, and run Oracle Application Development Framework (ADF) applications.

This section contains the following sections:

- [Getting Started with Oracle ADF](#)
- [Working with the Oracle ADF Model Layer](#)
- [Working with Oracle ADF Controller](#)
- [Working with Oracle ADF Faces](#)
- [Deploying an Oracle ADF Application](#)
- [Debugging an Oracle ADF Application](#)
- [Using AppXray for Oracle ADF Artifacts](#)
- [Refactoring Oracle ADF Components](#)
- [Reusing Oracle ADF Application Components](#)
- [Configuring and Using ADF with GlassFish Server](#)
- [Appendix A Oracle ADF XML Files](#)

2.1 Getting Started with Oracle ADF

The Oracle Application Development Framework (Oracle ADF) is an end-to-end application framework that builds on Java Platform, Enterprise Edition (Java EE) standards and open-source technologies. You can use Oracle ADF to implement enterprise solutions that search, display, create, modify, and validate data using web, wireless, desktop, or web services interfaces. Because of its declarative nature, Oracle ADF simplifies and accelerates development by allowing users to focus on the logic of application creation rather than coding details.

Note: This document contains links to more detailed conceptual information in other Oracle ADF documentation. Keep in mind that while the information is sometimes presented in the context of another Oracle IDE (JDeveloper), the concepts are applicable to OEPE as well. For your ease of understanding, equivalent JDeveloper and OEPE IDE components are listed in the table below.

Oracle JDeveloper IDE Component	Corresponding OEPE IDE Component
Applications Window	Project Explorer
Structures Window	Outline
Components Window	Palette
Data Controls Panel	Data Palette
Properties Window	Properties
Databases Window	Data Source Explorer
Log Window	Console
Application Servers Window	Servers

2.1.1 Understanding the Oracle ADF Architecture

In line with community best practices, applications you build using the ADF web technology stack achieve a clean separation of business logic, page navigation, and user interface by adhering to a model-view-controller architecture.

For more information about the Oracle ADF Architecture, see the Oracle ADF Architecture section in the "Introduction to Building Fusion Web Applications with Oracle ADF" chapter of *Developing Fusion Web Applications with Oracle Application Development Framework*.


2.1.2 Configuring Oracle WebLogic Server

Before starting to create your pages, you need to create a server configuration for Oracle WebLogic Server (WLS). It will be the link to the WLS instance that will be used to run the project.

Note: To use GlassFish ADF enabled runtime with you JAVA or ADF application see [Section 2.10, "Configuring and Using ADF with GlassFish Server"](#).

The WebLogic server should be running. You can use an existing domain or create a new domain.

To configure WLS using an existing domain:

1. From the main menu, select **Window > Show View > Other**.
2. In the Show View dialog, select **Server > Servers** to open the Servers pane.
3. In the Servers pane, right-click and select **New > Server**.
Alternatively if there are no servers defined, click the [No servers are available. Click this link to create new server. link](#).
4. On the Define a New Server page of the New Server wizard, expand **Oracle** and select the type of Oracle WebLogic server you are running, for example, **Oracle WebLogic Server 12c (12.1.3)**. Enter the server's hostname, for example localhost and click **Next**.
5. On the server details page of the Define a New Server wizard, click **Browse**  next to **WebLogic Home**. In the Browse for Folder dialog navigate to the WebLogic home directory and click **OK**.

Alternatively, if you have already defined domains for the IDE they are listed when you choose **Browse**  > **Known Domains**.

Click **Next**.

6. On the Add and Remove page, in **Available** select your application and click **Add** to shuttle the selection to the **Configured**. This associates your application with the newly created domain.


Click **Finish**.

The new server connection appears in the Servers pane. Note that the server is in a stopped state.

To configure WLS and create a new domain:

1. From the main menu, select **Window > Show View > Other**.
2. In the Show View dialog, select **Server > Servers** to open the Servers pane.
3. In the Servers pane, right-click and select **New > Server**.

Alternatively if there are no servers defined, click the `No servers are available. Click this link to create new server. link`.

4. In the New Server wizard on the Define a New Server page, select the server type. Select **Oracle** and then select the type of Oracle WebLogic server you are running, for example, **Oracle WebLogic Server 12c (12.1.3)**. Click **Next**.
5. To create a new domain, in the server details page of the Define a New Server wizard, click **Create**  then **Create Domain**. In the New WebLogic Domain dialog:
 - Enter a name for the domain.
 - Optionally specify a password that will override the default password `welcome1`.
 - Check that the extensions you want are selected.

The IDE validates the domain path you choose and automatically enters the Java home path in **Java home**. It also lists the relevant extensions on the server.

Click **Next**.

6. In the New WebLogic Domain dialog, specify a name and a location for the domain. Optionally, you can specify a new password (the default is `welcome1`). Under **Extensions**, select **Oracle JRF - 12.1.3.0**.

Click **Finish**.


7. On the New Server wizard, **Domain directory** is now populated. Click **Next**.
8. On the Add and Remove page, in the **Available** list, select your application. Then click **Add** to shuttle the selection to the Configured pane. This associates your application with the newly created domain.

Click **Finish**.


The new server connection appears in the Servers pane. Note that the server is in a stopped state.

To configure WLS using the WebLogic Server configuration wizard:

1. From the main menu, select **Window > Show View > Other**.
2. In the Show View dialog, select **Server > Servers** to open the Servers pane.

3. In the Servers pane, right-click and select **New > Server**.
Alternatively if there are no servers defined, click the `No servers are available.` Click this link to create new server. link.
4. In the New Server wizard on the Define a New Server page, select the server type. Select **Oracle** and then select the type of Oracle WebLogic server you are running, for example, **Oracle WebLogic Server 12c (12.1.3)**. Click **Next**.
5. To create a new domain or update an existing domain using the WebLogic Server configuration wizard, click **Create**  then **Launch Domain Configuration Wizard**.

The Oracle Fusion Middleware Configuration Wizard opens where you can create a new domain or update an existing domain.

Once you have specified the domain in the configuration wizard, click **Browse**  next to **WebLogic Home**. In the Browse for Folder dialog navigate to the WebLogic home directory and click **OK**.

6. On the New Server wizard, **Domain directory** is now populated. Click **Next**.
7. On the Add and Remove page, in the **Available** list, select your application. Then click **Add** to shuttle the selection to the Configured pane. This associates your application with the newly created domain.

Click **Finish**.

The new server connection appears in the Servers pane. Note that the server is in a stopped state.


2.1.3 Creating an Oracle ADF Application

OEPE supports the creation of the following ADF application artifacts:

- JPA Entity
- EJB Session Bean as service
- JSF Managed Bean for binding
- ADF Data Controls for binding data to the View and Controller
- ADF Task Flow as Controller
- ADF Components as View

To create an ADF application:

1. From the main menu, select **File > New > ADF Application**.
2. In the New Oracle ADF Application dialog, enter the following:
 - Application name
 - Application project location The default is `user_home/workspace`.
 - Dynamic web project name. The default is `Web`. This name is appended to the application name.
3. Click **New Runtime** to set the WebLogic Runtime environment. The New Server Runtime Environment wizard opens.
4. In the New Server Runtime Environment wizard, Select **Oracle** and then select the type of Oracle WebLogic server you are running, for example, **Oracle WebLogic Server 12c (12.1.3)** and click **Next**.

5. On the server details page, click **Browse**  next to **WebLogic home** and browse to the location of your WebLogic server installation. Click **OK**.
6. In the New Oracle ADF Application dialog, select a JPA project in the **JPA Project** field. To optionally create a new JPA project, click **New JPA Project**.
7. In the New JPA Project wizard, enter the following details then click **Next**:
 - Project name
 - Project location (*user_home/workspace* by default)
 - Target runtime
 - JPA version (If you have installed JPA 2.0, the version defaults to 2.0)
 - Configuration

Note: You can modify the default configuration to add the Oracle WebLogic EJB Extensions facet or the Oracle WebLogic Utility Module Extensions facet. These facets cannot be added together. The benefit of using the WLS Extensions facet during project creation is that you can take advantage of the EclipseLink/TopLink shared libraries that ship with Oracle WebLogic Server. If you do not select a WLS Extensions facet, you will need to manually configure your JPA runtime.

- EAR Membership. Select an EAR file to add the project to. Note that the JPA project will be added to the application's EAR file by default.
 - Working sets. For more information, see the help topic "Working Sets" in the Workbench User Guide > Concepts > Workbench.
8. On the Java page, specify the source folders on build path and the default output folder, and click **Next**.
 9. On the JPA Facet page, specify the **Platform**, **JPA Implementation**, and database connection settings, and click **Finish**.
 10. In the New Oracle ADF Application dialog, you will see that your JPA project is listed in the **JPA Project** field. Click **Add project to working sets** if you would like to include the current project in a working set. Select an existing working set from the **Working sets** menu or create a new one.
 11. Click **Finish**.

Two projects are created in the Project Explorer - an EAR project and a dynamic web project. If you created a JPA project, that is also added to the Project Explorer.

Note: Using the ADF application wizard is not the only way you can create ADF applications. You can also add the ADF facets to your existing project to use ADF design time features.

2.2 Working with the Oracle ADF Model Layer

The model layer represents the data values related to the current page.

2.2.1 Creating the JPA Model Project

If you did not create the JPA project when you created your ADF Application, you can create one using the New Gallery.

To create the JPA project:

1. From the main menu, select **File > New > Other**. From the New Gallery, open the **JPA** node and select **JPA Project**.
2. In the New JPA Project wizard, enter the following details:
 - Project name.
 - Project location. By default, this is `user_home/workspace`.
 - Target runtime. Choose from the list of those defined, or click **New Runtime** to define a new one.
 - JPA version. The default is 2.1 and the other available JPA versions are 1.0 and 2.0.
 - Configuration. The default is **Basic JPA Configuration**. You can modify it by clicking **Modify** and choosing to enable additional facets. The best option here depends on how you separate your EAR modules. If you store JPA entities as EJBs, then the EJB model is a better choice because it configures the project to use JPA (creating the `persistence.xml`, adding the design time tools for JPA like entity creation and annotation content assist) as well as EJB (wizards to create a new session bean). If you are separating the entities and EJBs into different modules, then it makes sense to use the JPA configuration preset and use the current preset only for Entity configuration, then create an additional EJB utility module later.

Note: You can modify the default configuration to add the Oracle WebLogic EJB Extensions facet or the Oracle WebLogic Utility Module Extensions facet. These facets cannot be both added to the same configuration. The benefit of using the WLS Extensions facet during project creation is that you can take advantage of the EclipseLink/TopLink shared libraries that ship with Oracle WebLogic Server. If you do not select a WLS Extensions facet, you will need to manually configure your JPA runtime.

- EAR Membership. Select an EAR file to add the project to. If you do not select an EAR, the project is added to the current ADF EAR project.
 - Working sets
3. Once you have added details in the JPA Project page of the New JPA Project Wizard, click **Next**.
 4. On the Java page, specify the source folders on build path and the default output folder, and click **Next**.
 5. On the JPA Facet page, specify the Platform, JPA Implementation, and database connection settings, and click **Finish**.

2.2.2 Creating JPA Entities from Tables

You create JPA Entities from tables.

To create JPA Entities from Tables:

1. Right-click your JPA project, and in the context menu, select **JPA Tools > Generate Entities from Tables**.
2. On the Generate Custom Entities wizard - Select Tables page, select a database connection from the **Connection** dropdown list, or click the **New Connections** button to add a connection.
3. On the Generate Custom Entities wizard - Select Tables page, select a database schema from the **Schema** dropdown list, and select the tables you want to generate entities from in the **Tables** area. Then click **Next**.
4. On the Table Associations page, review and optionally edit the table associations. Then click **Next**.
5. On the Customize Default Entity Generation page, optionally customize aspects of entities that will be generated by default from database tables. You must specify a Java package.
6. On the Customize Individual Entities page, optionally customize individual entities. Then click **Finish**.
7. In the Project Explorer, expand the `src` folder in your JPA project to see the JPA entities that have been created.

2.2.3 Working with Session Beans

Session beans implement business logic. A session bean instance serves one client at a time.

2.2.3.1 Generating a Session Bean on Selected JPA Entities

You can generate session beans on JPA entities.

To generate a session bean on selected JPA entities:

1. Right-click your JPA project to open the context menu.
2. In the context menu, select **New > Session Bean (EJB 3.x) from JPA Entities**.
3. On the Session Bean (EJB 3.x) page of the Create EJB 3.x Session Bean from JPA Entities wizard, specify a Java package and a class name. Then click **Next**.
4. On the Select Entities page, indicate which entities to access via the generated session bean. Then click **Finish**.
5. You can view the generated session bean in the Project Explorer.

2.2.3.2 Generating a JSF Managed Bean

You can generate a JSF Managed Bean which is a wrapper on a session bean.

To generate a JSF Managed Bean:

1. Right-click the session bean in the Project Explorer to open the context menu.
2. In the context menu, select **New > Data Model Components > Generate JSF Managed Bean**.
3. On the Managed Bean page of the Create Managed Bean wizard, specify the **Java package** and the **Session bean** to wrap. Then click **Finish**.
4. View the generated JSF Managed Bean in the Project Explorer.

-
-
- Notes:**
- While regeneration is available for EJBs, it is currently not supported for a Managed Bean. The Create Managed Bean wizard will fail if the class already exists.
 - OEPE uses the `@generated` annotation to determine what can be overwritten when regenerating the EJB. If you customize one of the generated methods, persist for example, you will need to remove the `@generated` label from the method before regenerating an EJB.
-
-

2.2.3.3 Generating a Session Bean and a JSF Managed Bean using the Data Components Model Wizard

You can generate a Session Bean facade on selected entities and generate a JSF Managed Bean to wrap the services in the session facade using the Data Components Model Wizard.

To generate a Session Bean and a JSF Managed Bean using the Data Components Model Wizard:

1. Open the New Gallery by selecting **File > New > Other**.
2. Expand **Application Development Framework**, and double-click **Data Model Components**.
3. On the Session Bean (EJB 3.x) page of the Create Data Model Components wizard, specify the Java package and class name for the session bean. Then click **Next**.
4. On the Select Entities page, indicate which entities to access via the generated session bean, then click **Next**.
5. On the Data Object page, select **JSF Managed Bean** or **ADF Data Control** as the Data object type. Then specify name and location details for the Managed Bean and click **Finish**. For more information on ADF Model Data Binding see "[Working with ADF Model Data Binding](#)".
6. In the Project Explorer, view the generated session bean facade in the JPA project and the JSF Managed Bean in the web project.

2.2.3.4 Editing a Session Bean

You can edit sessions beans.

To edit the session bean facade:

1. Right-click the session bean to open the context menu.
2. In the context menu, select **Data Model Components > Edit Session Bean Facade**.
3. On the Select Entities page, indicate which entities you want to add to or remove from the session bean facade.

2.2.4 Working with ADF Model Data Binding

ADF Model implements two concepts that enable the decoupling of the user interface technology from the business service implementation: data controls and bindings. Data controls abstract the implementation technology by using standard metadata interfaces to describe the bean's operations and data collections, including information about the properties, methods, and types involved. In OEPE, you can view this information in the Data Palette, and create databound HTML elements for JSP pages by dragging and dropping data control artifacts from the Data Palette onto the editor

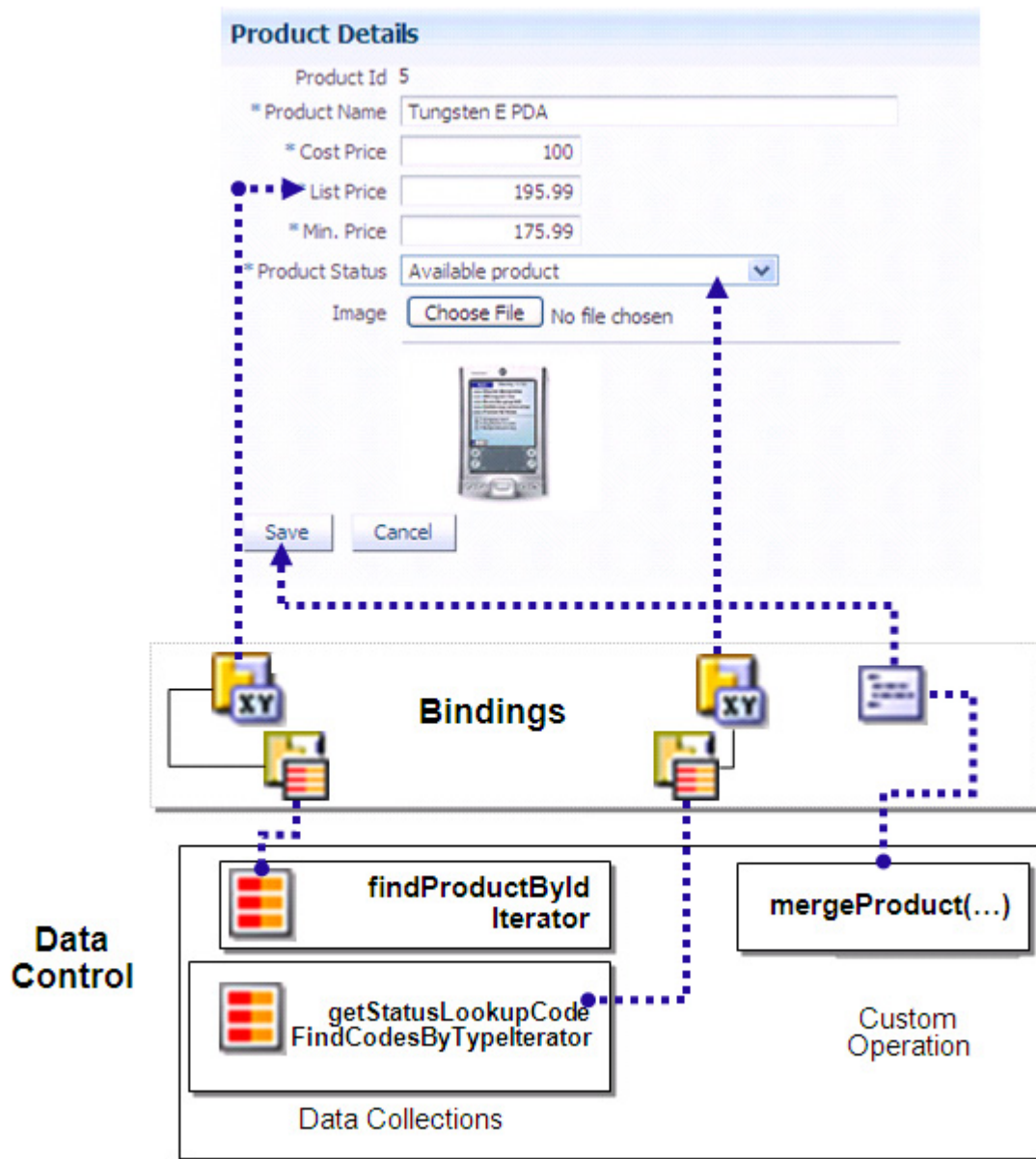
for a page. OEPE automatically creates the metadata that describes the bindings from the page to the services. At runtime, the ADF Model layer reads the metadata information from the appropriate XML files for both the data controls and the bindings, and then implements the two-way connection between your user interface and your services.

Declarative bindings abstract the details of accessing data from data collections in a data control and of invoking its operations. There are three basic kinds of declarative binding objects:

- Executable bindings: Include iterator bindings, which simplify the building of user interfaces that allow scrolling and paging through collections of data and drilling-down from summary to detail information. Executable bindings also include bindings that allow searching and nesting a series of pages within another page.
- Value bindings: Used by UI components that display data. Value bindings range from the most basic variety that work with a simple text field to more sophisticated list and tree bindings that support the additional needs of list, table, and tree UI controls.
- Action bindings: Used by UI command components like hyperlinks or buttons to invoke built-in or custom operations on data collections or a data control without writing code.

[Figure 2-1, "ADF Bindings"](#) shows how bindings connect UI components to data control collections and methods.

Figure 2-1 ADF Bindings



The group of bindings supporting the UI components on a page are described in a page-specific XML file called the page definition file. The ADF Model layer uses this file at runtime to instantiate the page's bindings. These bindings are held in a request-scoped map called the binding container.

2.2.4.1 Creating ADF Data Controls

Once you have your application's services in place, you can use OEPE to create data controls that provide the information needed to declaratively bind UI components to those services. In an ADF application, you can create a data control for the session bean or POJO, and that data control will contain representation of all data on the bean. The data control consists of a number of XML metadata files that define the capabilities of the service that the bindings can work with at runtime.

To create a data control from the Project Explorer:

1. In the Project Explorer, right-click the session bean or POJO for which you want to create a data control.
2. From the context menu, select **Model Components > Create ADF Data Control**.

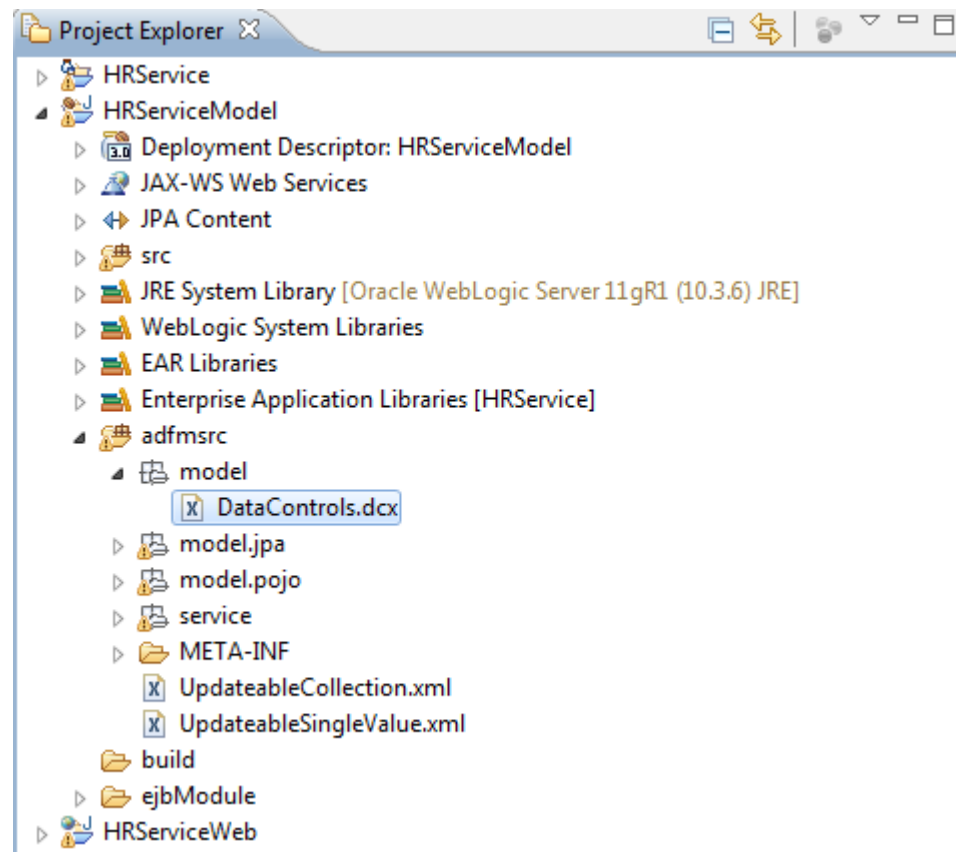
Alternatively, you can also create a data control from the New Gallery.

To create a data control from the New Gallery:

1. Right-click your model project and select **New > Other** from the context menu.
2. In the New gallery, choose **Oracle > Application Development Framework > ADF Data Control**.
3. On the Bean Class page, select project [**Project**] and the bean [**Bean Class**] from which to create the data control. Note: The Project dropdown list contains all eligible projects in the workspace.
4. Preview and confirm the project changes on the Summary page and click **Finish**.

The metadata file `datacontrols.dcx` is created in the `adfmsrc > model` node of your Model project, as shown in [Figure 2-2, "Data Control File in Project Explorer"](#). When you create a data control based on an EJB session bean, the data control contains a representation of all the methods exposed on the bean, as well as underlying entity beans, and the methods and properties exposed on those.

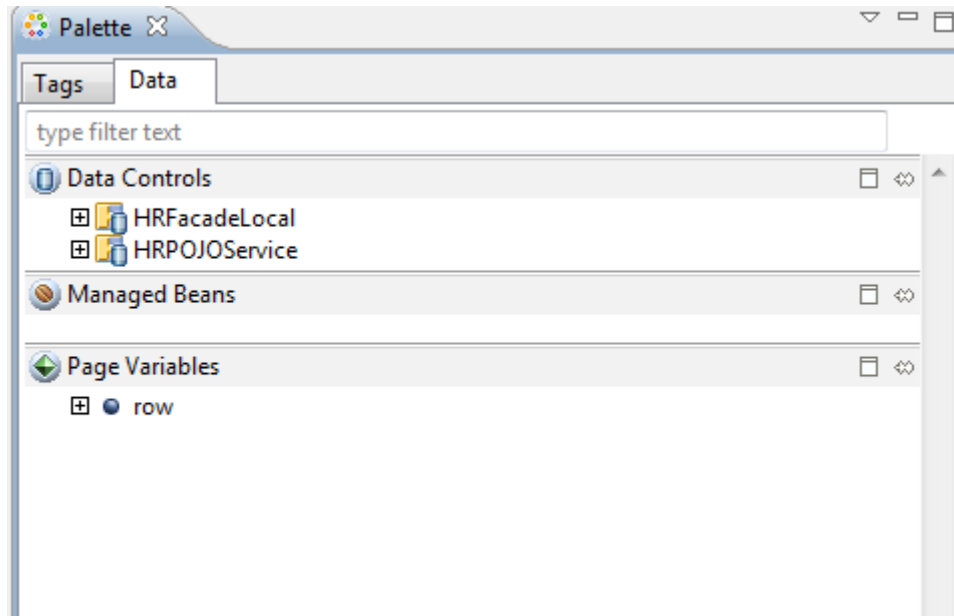
Figure 2-2 Data Control File in Project Explorer



For a description of Oracle ADF Data Bindings metadata files and links to more information, see [Section 2.11, "Appendix A Oracle ADF XML Files."](#)

With a web page in focus, you can open the Data section of the Palette to view and drag-and-drop data controls, JSF Managed Beans, and page variables on to the page, shown in [Figure 2-3, "Data Palette"](#).

Figure 2-3 Data Palette



For more information about ADF data controls, see "Using ADF Data Controls" in *Developing Applications with Oracle ADF Data Control*.

2.2.4.2 Using the ADF Data Control Manager

The Data Control Manager shows all the data controls available to your ADF project. Use the Data Control Manager to view details about your data control objects, including the data type associated with your Java elements, and related artifacts.

The Data Controls tab of the Data Control Manager shows the data control tree. The Structure Definitions tab shows the structure definition control objects. Every data control object exposes the structure definition of your data objects and is shown in the Structure Definitions view. When you are customizing your data structure in the 12c runtime, you customize the structure definition object. Customizing a structure definition object impacts every data control object that exposes that structure definition object.

Note that in 12c applications you can customize your data structure definition object, but in 11g applications, the Data Control Manager shows your available data controls, java elements, and object types, but doesn't allow customizing of the structure definition object.

There are a three ways to access the Data Control Manager:

1. Opening a data control .dcx file.
2. When creating a data control, the last page of the wizard has a check box that you can select to open the data control manager.
3. If you are editing a ADF page, you can open the manager through the Data Palette view: **Palette > Data tab > right-click the control > Data Control Manager**.

Figure 2–4 shows the Data Control Manager with the **Show structure definition** link available. When you select a data control object, with an editable structure definition, this link changes to **Edit structure definition** as shown in Figure 2–5.

Clicking on the **Edit structure definition** link selects the appropriate structure definition on the second tab of the Data Control Manager, as shown in Figure 2–6. In this dialog, you can enter custom label and tool tip text used by a widget that is rendering the selected data control object on a ADF web page. You can also create, edit, or delete a list of values for the structure definition, as shown in. Use the list of values to determine which values are available for the selected attribute defined by either a collection accessor or an attribute of a collection accessor.

In the web page editor, when you select from a list of available widgets for an attribute of a data control, OEPE determines whether the list of values can be used at runtime. If yes, it displays the name of the list of values in the Edit List Binding dialog, as shown in Figure 2–7.

Figure 2–4 Data Control Manager - Show Structure Definition

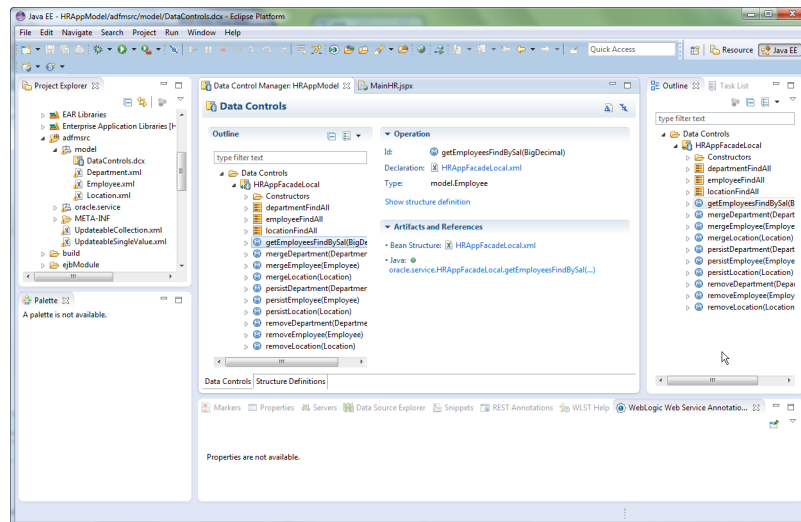


Figure 2–5 Data Control Manager - Edit Structure Definition

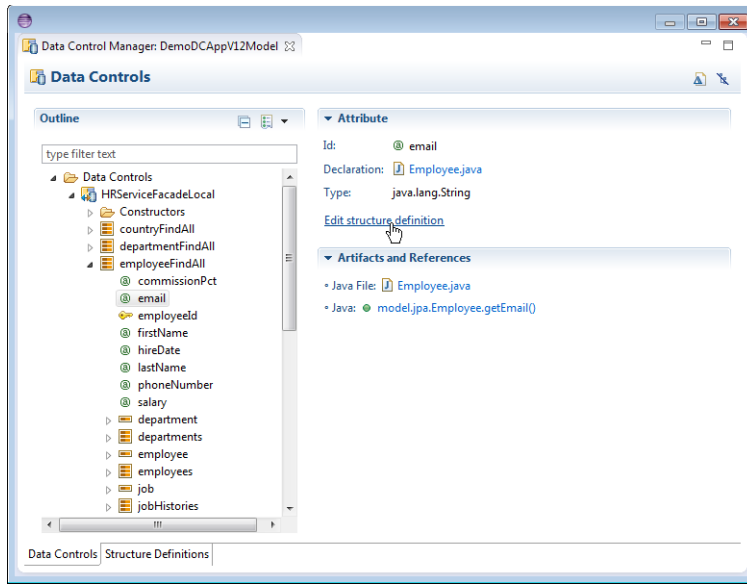


Figure 2–6 Data Control Manager - Edit List of Values

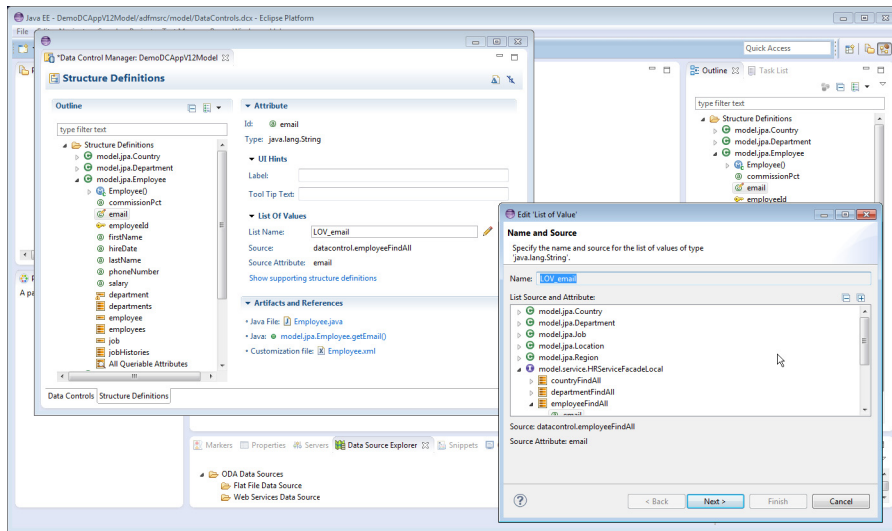
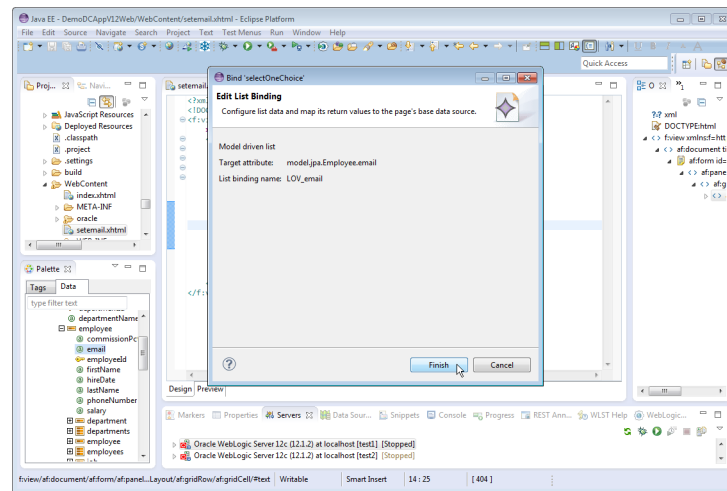


Figure 2–7 Data Control Manager - Edit List Binding

2.2.4.3 Using the Data Palette to Create UI Components

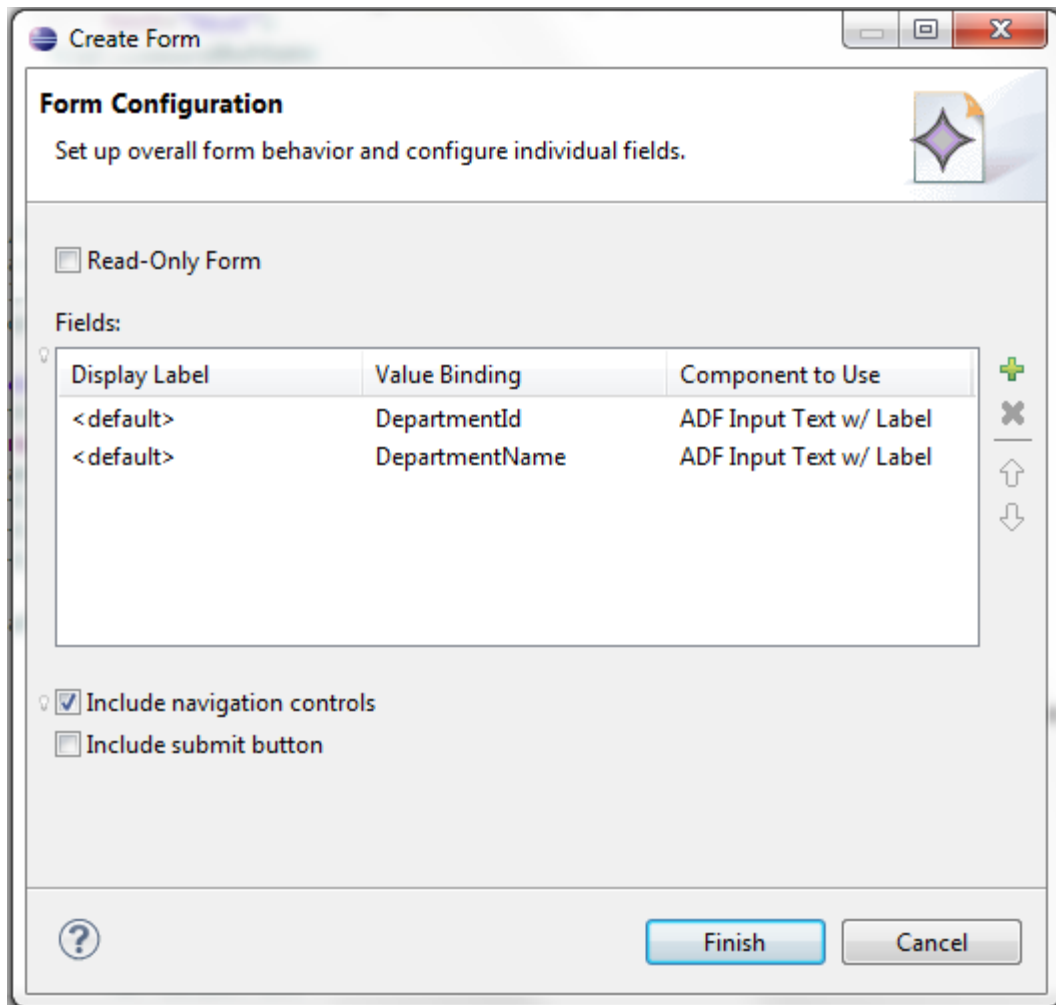
OEPE provides you with a predefined set of UI components from which to choose for each data control item you drop.

To use the Data Palette to create UI components:

1. Select an item in the Data Controls panel and drag it onto the editor for your page. You can also drag it and drop it from the data palette to the Outline view.
2. From the ensuing context menu, select a UI component. When you drag an item from the Data palette and drop it on a page or task flow, OEPE displays a context menu of all the default UI components available. The UI components currently supported are: **Form**, **Graph**, **Navigation**, **Single Selection**, and **Table**.

Depending on the component you select from the context menu, OEPE displays a dialog that enables you to define how you want the component to look.

For example, if you select **Form > ADF Form**, the Create Form dialog is opened, as shown in [Figure 2–8, "Create Form Dialog"](#). You can specify if you want it to be a **Read-Only Form**. Additionally, you can edit the Fields table to change the **Display Label**, **Value Binding**, and **Component to Use** elements. Additionally, you can add or delete value bindings, and specify if you want to include navigation controls and a submit button.

Figure 2–8 Create Form Dialog

The resulting code appears in the source editor for your page.

3. You can click the **Edit Component Definition** toolbar button on the Properties window to edit the properties of your UI component.

Note: This feature is currently supported only for ADF Table and ADF DVT Graph components.

For more information using the data palette, see "Using ADF Data Controls" in *Developing Applications with Oracle ADF Data Control*.

To run the ADF page containing your newly added component, right-click the page in the Project Explorer and select Run As > Run on Server. For more information, see [Section 2.5, "Deploying an Oracle ADF Application."](#)

Note: When running an ADF page from within OEPE, graph components may render inconsistently in the Eclipse browser. Using an external browser is recommended for best results.

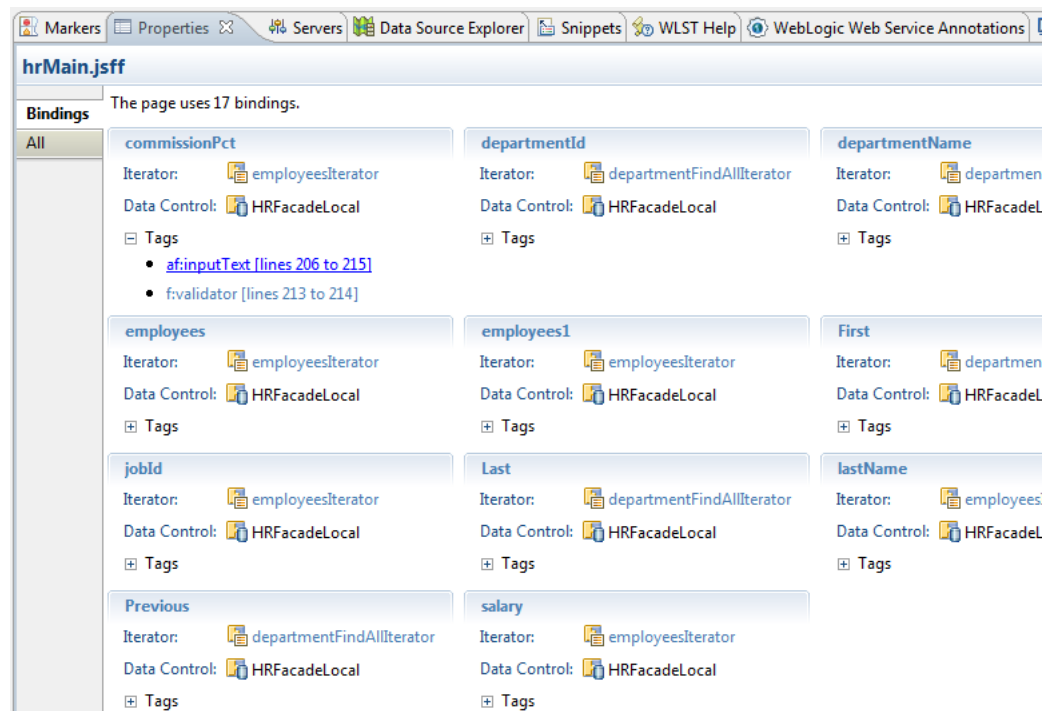
2.2.4.4 Using the Bindings Tab in the Properties Window

You can view the bindings associated with a page in the Bindings tab of the Properties window. The tab provides contextual bindings information based on the tag currently in focus in the editor, as shown in [Figure 2–9, "Bindings tab in Properties Window"](#).

The Bindings tab provides the following features:

- If you place the cursor in the `jsp:root` tag of the page, the Bindings tab provides an overview of all the bindings used in the page.
- Each binding is listed along with its iterator and associated data control, in addition to the tags it is used in.
- Click on the binding id or iterator to open the item in the page definition editor.
- Click on a tag to highlight that portion of the source code in the web page editor.

Figure 2–9 Bindings tab in Properties Window



2.2.4.5 Working with Page Definition Files

Page definition files define the binding objects that populate the data in UI components at runtime. For every page that has ADF bindings, there must be a corresponding page definition file that defines the binding objects used by that page. Page definition files provide design time access to all the ADF bindings. At runtime, the binding objects defined by a page definition file are instantiated in a binding container, which is the runtime instance of the page definition file.

When you drag and drop an item from the Data Palette to the page, OEPE automatically creates a page definition file for that page and adds definitions for each binding object referenced by the component. For each subsequent databound component you add to the page, OEPE automatically adds the necessary binding object definitions to the page definition file. By default, the page definition files are located in the `view.PageDefs` package in the `adfmsrc` directory of the web project.

OEPE names the page definition files using the following convention:

`pageNamePageDef.xml`

where `pageName` is the name of the JSP page. For example, if the JSF page is named `home.jsp`, the default page definition file name is `homePageDef.xml`.

2.2.4.6 Opening a Page Definition File in the Page Definition Editor

You can open a page definition file in the page definition editor using any of the following ways:

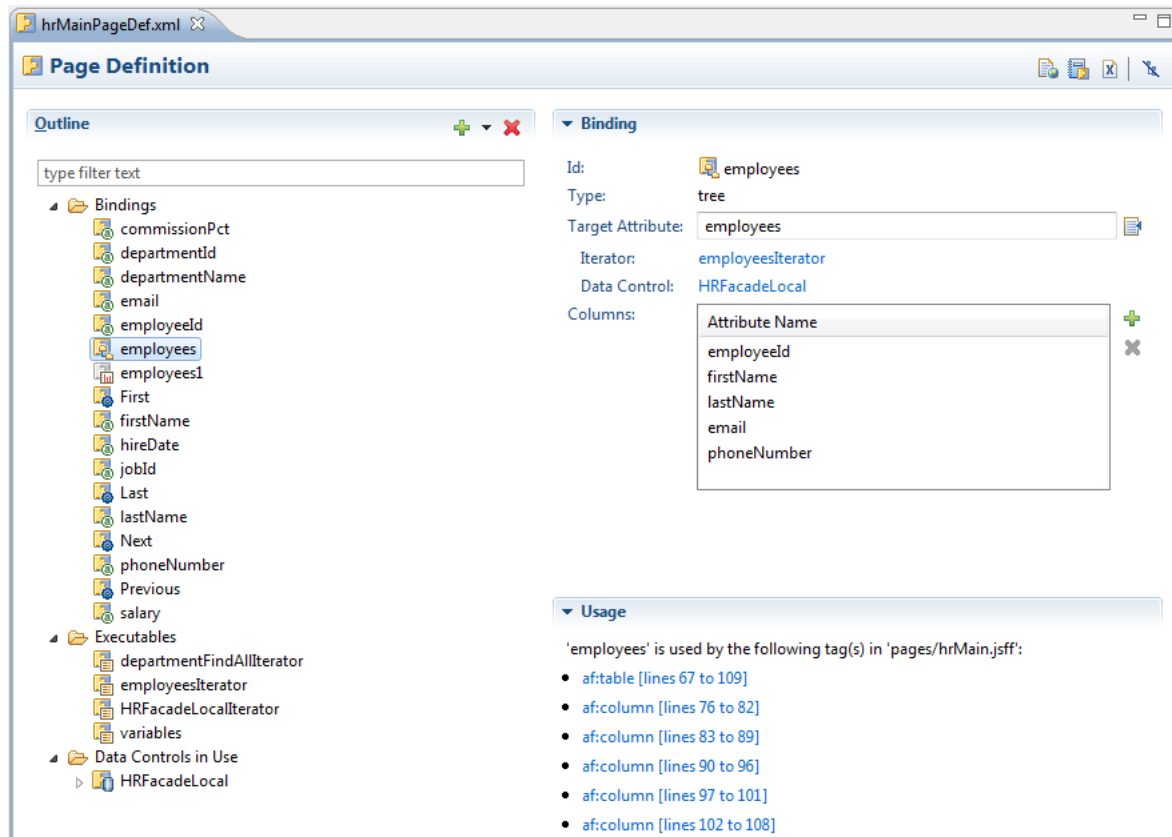
- Right-click the page in the Project Explorer and select **Open Page Definition**. If the page does not have an associated page definition file, OEPE asks if you want to create it.
- With a page active in the editor, from the main menu, select **Navigate > Go To > Page Definition**.
- In the Bindings tab of the Properties view, click on a binding id to open it in the page definition editor.
- If the page definition file already exists, you can double-click it in the Project Explorer.

2.2.4.7 Understanding the Page Definition Editor

The page definition editor allows you to view and configure bindings, as shown in [Figure 2–10, "Page Definition Editor"](#). It consists of the following sections:





- **Outline:** Shows three different types of objects: bindings, executables, and the data controls in use. You can also add a new binding or executable using the New button (green plus icon). When you click an item in the Outline section, its corresponding details are shown in the right pane in the Binding (for bindings) and Iterator Binding (for executables) section.
- **Binding:** Displays information about the binding currently selected in the Outline section.

Figure 2–10 Page Definition Editor



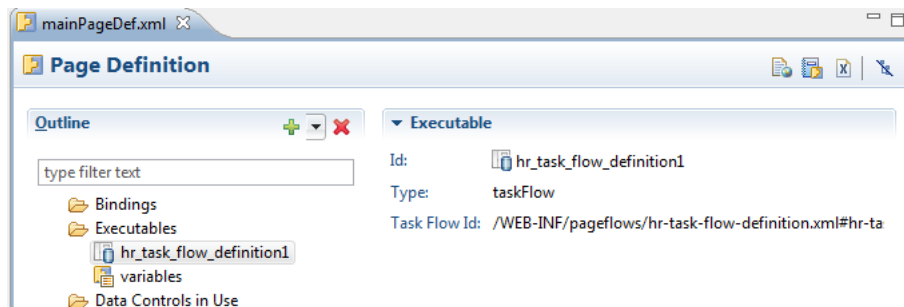
- **Executable:** Displays information about the executable currently selected in the Outline section.
- **Iterator Binding:** Displays information about the currently selected iterator in the Outline section.
- **Usage:** Displays usage information. For a binding, it displays the tags that use it. For an executable, it displays the bindings that use it. For a data control, the bindings and executables that use it are displayed.
- **Bindings Summary or Executable Summary, Validation Details, Usage Details:** These sections display summary information when the parent Bindings or Executables folder is selected in the Outline section.
- **Data Control:** Displays the Id and Type of the currently selected data control.
- **Data Control Child:** Displays the Id and Type of the currently selected Data Control child element.

The toolbar icons on the top right corner of the page definition editor are:

-  **Open *file_name*:** Click to open the associated JSP page, where *file_name* is the name of the file.
-  **Open 'adfmsrc/view/DataBindings.cpx':** Click to open the DataBindings.cpx file.
-  **Open this page definition using the XML editor**
-  **Hide Outline:** Click to hide the Outline section.

The page definition editor is tolerant of artifacts that are not supported. For example, in [Figure 2–11, "Unsupported Artifact"](#), the editor recognizes the tasteful executable even though the artifact is not supported.

Figure 2–11 Unsupported Artifact



If a page definition file is modified outside of the page definition editor, the editor will inform you via the External Changes Detected dialog. The contents of the file are then reloaded. Additionally, the editor also supports Forward and Back navigation from the Eclipse IDE. You can navigate previous selections using either the left and right arrows on the Eclipse toolbar or the main menu items **Navigate > Back and Navigate > Forward**.

2.2.4.8 Understanding Bindings and Executables

Declarative bindings abstract the details of accessing data from data collections in a data control and of invoking its operations.

Bindings

There are three types of Bindings binding objects used to bind UI components to objects on the data control:

- **Value:** Displays data in UI components by referencing an iterator binding. Each discrete UI component on a page that will display data from the data control is bound to a value binding object. Value binding objects include:
 - **Attribute Values:** Binds text fields to a specific attribute in an object (also referred to as an attribute binding object.)
 - **List:** Binds the list items to all values of an attribute in a data collection.
 - **Tree:** Binds an entire table to a data collection and can also bind the root node of a tree to a data collection.
 - **Button (boolean):** Binds a checkbox to a boolean value for an attribute.
- **Method Action:** Binds command components, such as buttons or links, to custom methods on the data control. A method action binding object encapsulates the details about how to invoke a method and what parameters (if any) the method is expecting.
- **Action:** Binds command components, such as buttons or links, to built-in data control operations (such as, Commit or Rollback) or to built-in collection-level operations (such as, Create, Delete, Next, or Previous).
- **Task Flow:** A binding used to encapsulate a reusable portion or region of an application. You can isolate a small, specific piece of application functionality as a region that can be reused throughout the application. Task flow binding enables you to extract, parameterize, and package

There are five types of supported executable binding objects:

- **Iterator:** Binds to an iterator that iterates over view object collections. There is one iterator binding for each collection used on the page. All of the value bindings on the page must refer to an iterator binding in order for the component values to be populated with data at runtime.

When you drop a collection or an attribute of a collection on the page, an iterator binding is automatically added as an executable. Iterator binding objects bind to an underlying ADF `RowSetIterator` object, which manages the current object and current range information. The iterator binding exposes the current object and range state to the other binding objects used by the page. The iterator range represents the current set of objects to be displayed on the page. The maximum number of objects in the current range is defined in the `rangeSize` attribute of the iterator. For example, if a collection in the data control contains products and the iterator range size is 25, the first 25 products in the collection are displayed on the page. If the user scrolls down, the next set of 25 is displayed, and so on. If the user scrolls up, the previous set of 25 is displayed. If your view object uses range paging, then you can configure the iterator binding to return a set of ranges at one time.

Note: If you have two pages each with an iterator binding bound to the iterator on the same view object (which you will if you drop the same collection, for example, on two different pages), then you should ensure that the `rangeSize` attribute is the same for both pages' iterator bindings. If not, the page with a smaller range size may cause the iterator to re-execute, causing unexpected results on the other page.

- **Method Iterator:** Binds to an iterator that iterates over the collections returned by custom methods in the data control.

A method iterator binding is always related to a method action binding object. The method action binding encapsulates the details about how to invoke the method and what parameters (if any) the method is expecting. The method action binding is itself bound to the method iterator, which provides the data.

You will see method iterator executable binding objects only if you drop a method return collection or an attribute of a method return collection from a custom method on the data control.

- **Variable Iterator:** Binds to an iterator that exposes all the variables in the binding container to the other bindings. While there is an iterator binding for each collection, there is only one variable iterator binding for all variables used on the page. (The variable iterator is like an iterator pointing to a collection that contains only one data object whose attributes are the binding container variables.)

Page variables are local to the binding container and exist only while the binding container object exists. When you use a data control method (or an operation) that requires a parameter that is to be collected from the page, OEPE automatically defines a variable for the parameter in the page definition file. Attribute bindings can reference the page variables.

- **Accessor Iterator:** Iterates over detail collections returned by accessor methods. Accessor iterators are always related to a master iterator, which is a method iterator. The accessor iterator returns the detail objects related to the current object in the master (or method) iterator.

- **Invoke Action:** Binds to a method that invokes the operations or methods defined in action or method action bindings during any phase of the page lifecycle.

2.2.4.9 Adding Bindings and Executables

Bindings are added on the Page Definition page, which can be accessed by right-clicking your page in the Project Explorer, and choosing Open Page Definition. For information on tree bindings see "[Working with Tree Bindings](#)".

To add a new binding:

1. In the Outline section, click the **New** icon and then select **Binding**.
2. In the New Binding dialog, select the type of the new binding, for example, `list`, and enter the **Id**. Then click **OK**. The binding is added to the Outline section in the page definition editor.
3. In the editor, specify binding attributes for the new binding that you just created. For example, if you created a list binding, select the **Target Attribute** from the Edit List Binding dialog, as well as a **List Source Attribute**. You can also add or delete **Field Mappings**.

To add a new executable:

1. In the Outline section of the Page Definition editor, click the **New** icon and then select **Executable**.
2. In the New Executable dialog, select the type of the new executable, for example, `methodIterator`, and enter the **Id**. Then click **OK**. The executable is added to the Outline section in the page definition editor.
3. In the editor, specify attributes for the executable that you just created. For example, if you created a `methodIterator`, select a method action in the **Binds** field and specify a **Range Size**.

2.2.4.10 Working with Tree Bindings

Tree bindings are used to display a collection of data in forms requiring a nested data structure. For example, if you are displaying an organization chart, for the organization entities such as Department, Location, and Employee, a tree binding is used to display the parent and each child node using a unique type. So, if Department is the root entity type, you can bind and show attributes of Department, such as Department Name and ID, as well as show attributes for child entity types Location and Employee.

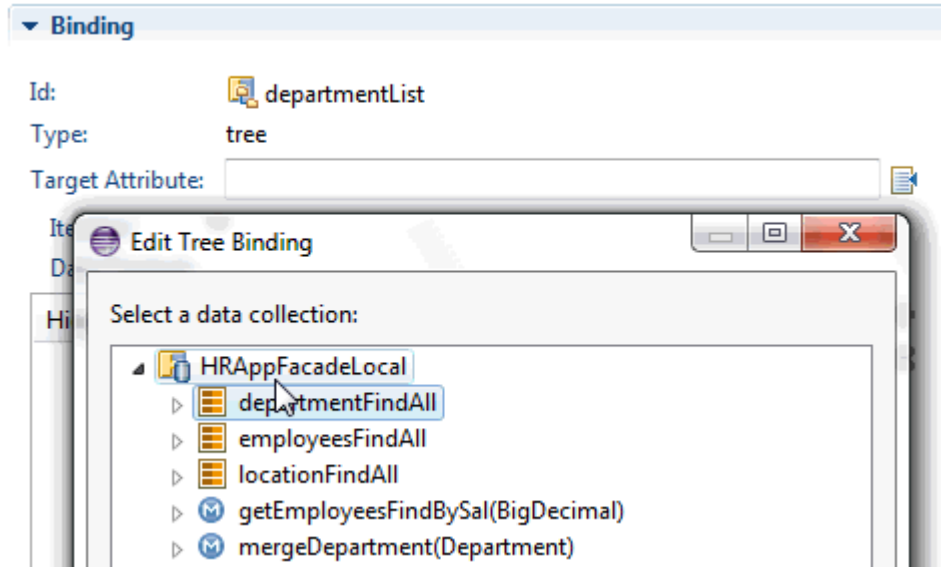
The system adds the ADF Tree component for a tree or hierarchical format, or the ADF Table component for a tabular display. For display such as organization charts or for any nested structure, the Hierarchy Viewer component is used.

To add a new tree binding:

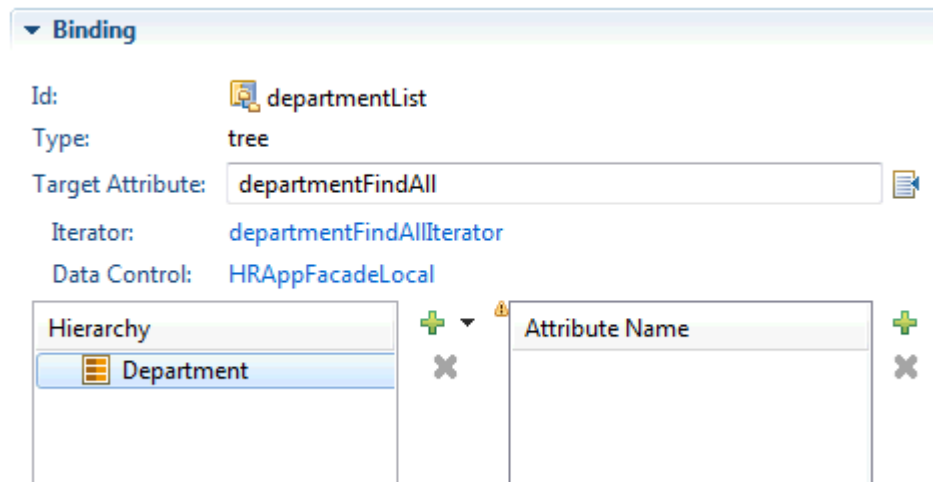
1. In the Outline section of the Page Definition editor, click the **New** icon and then select **Binding**.
2. In the New Binding dialog, select **Tree** and enter the ID. Click **OK**. The binding is added.

To add or change binding attributes for the new binding that you just created:

1. Add and edit attributes for new bindings on the right side of the Page Definition editor. Click on the page icon for Target Attribute. In the Edit Tree Binding dialog, select the data collection to bind. Click **OK**.



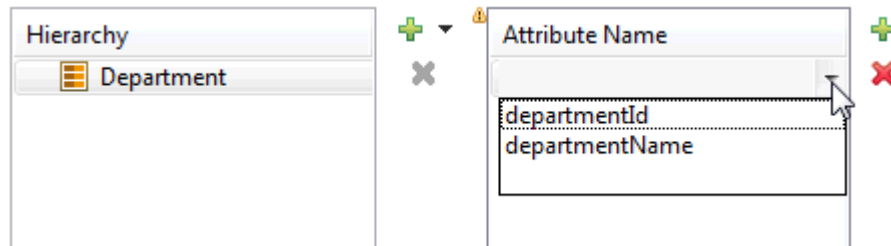
2. The editor displays the entity type of the data collection in the Hierarchy section on the left.



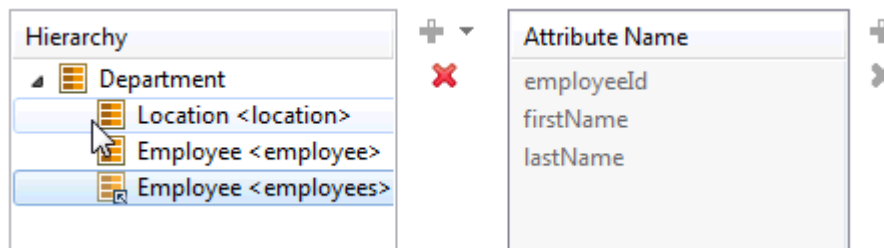
3. Click **Add (plus icon)** in the Attribute Name section on the right to select the attributes of the entity data type you added to the Hierarchy section.



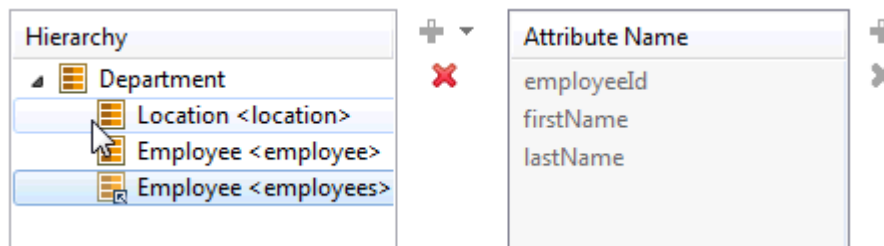
4. Double-click on the newAttribute entry and click on the drop-down icon to select the attribute.



5. To add a new entity type as a child of the selected node, Click **Add** in the Hierarchy section and select from the available type. Repeat steps 3-4 to select attributes of the type.



Note: There are two categories of nodes in the Hierarchy section. There are concrete categories which allow selection of attributes, and virtual categories which have been defined previously in the tree binding structure.



For more information about bindings and executables, see sections "Bindings Binding Objects" and "Executable Binding Objects" in the "Using ADF Model in a Fusion Web Application" chapter of *Developing Fusion Web Applications with Oracle Application Development Framework*.

2.2.5 Adding Data Binding to Existing UI Components

While the Data Palette enables you to design and create bound components in a single drag-and-drop action, in some cases, it may be preferable to create the basic UI components first and add the bindings later. For example, you can first create an ADF Faces component, and then bind it to the correct ADF control. Additionally, you can also rebind a UI component to a different data control.

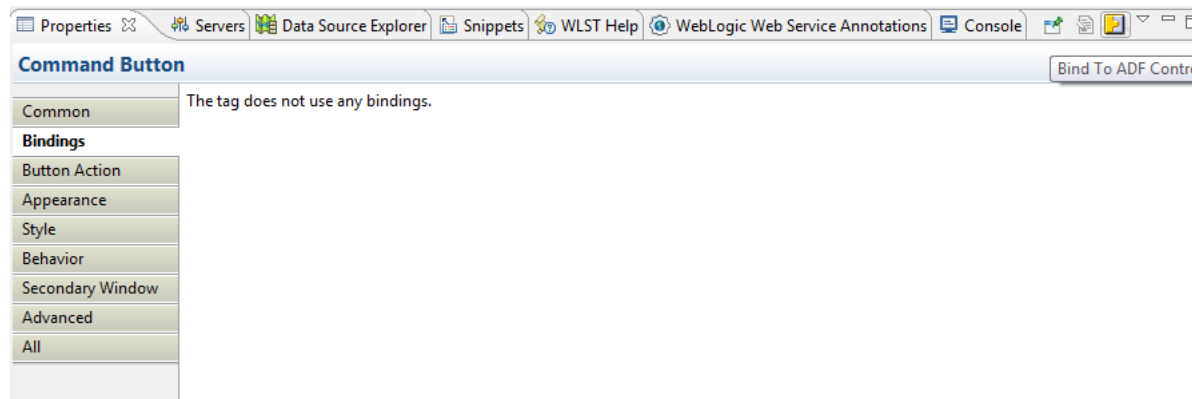
When designing web pages, keep in mind that ADF bindings can be added only to certain ADF Faces tags. The following lists the ADF Faces tags to which you can add ADF bindings.

- Text Fields

- af:inputText
- af:outputText
- af:outputLabel
- af:inputDate
- Tables
 - af:table
- Actions
 - af:commandButton
 - af:commandLink
 - af:commandMenuItem
 - af:commandToolbarButton
- Selection Lists
 - af:inputListOfValues
 - af:selectOneChoice
 - af:selectOneListbox
 - af:selectOneRadio
 - af:selectBooleanCheckbox

To apply ADF Model data binding to existing UI components:

1. In source editor for your page, select the UI component to which you want to add ADF bindings. When you select a component in the editor, OEPE simultaneously selects that component tag in the Outline window.
2. On the Properties pane, click the **Bind to ADF Control** toolbar button on the top right.



Note: Your project must already contain data controls for the Bind to ADF Control button to appear.

3. On the Select Data Control dialog, select an ADF Data Control or one of its properties. You must select a compatible control or property to continue. Click **OK**.

To rebind an existing UI component:

1. In source editor for your page, select the UI component you want to rebind to a different data control. When you select a component in the editor, OEPE simultaneously selects that component tag in the Outline window.
2. On the Properties pane, click the **Bind to ADF Control** toolbar button on the top right.
3. On the Select Data Control dialog, select the new data control or data control property you want to bind the UI component to. You must select a compatible control or property to continue. Click **OK**.

Depending on the UI component being rebound, OEPE displays a dialog where you can re-enter properties for the component. For example, if you are rebounding an ADF Table, the Edit Table dialog is displayed.

4. Specify properties for your component and click **Finish** to complete rebinding.

2.2.6 Debugging ADF Bindings

You can debug ADF bindings by setting breakpoints in the page definition editor. For more information, see [Section 2.6.4, "Setting and Using ADF Page Definition Breakpoints."](#)

2.2.6.1 ADF Page Definition Artifact Validation

OEPE provides real-time and on-demand validation of your page definition files even while the page definition editor is closed. Page definition validation warnings and errors are reported in the Markers view under the section ADF Page Definition Problems. You can set workspace preferences for ADF page definition validation from the Preferences dialog.

To specify preferences for ADF page definition validation:

1. From the main menu, select **Window > Preferences** to open the Preferences dialog.
2. In the Preferences dialog, select **ADF > Artifact Validation** to open the Artifact Validation page.
3. Specify validation options as per your preference:
 - **Disable validation:** Click to stop ADF page definition validation.
 - **Reduce all validation errors to warnings:** Selected by default. Deselect if you want validation issues to be reported as errors, thereby preventing deployment to WebLogic Server. This option is disabled if **Disable validation** is selected.
 - **Run Validation:** Click to run a workspace-wide validation based on current saved preferences. This option is disabled if Disable validation is selected.
 - On clicking **OK** or **Apply**:
 - If **Disable validation** is unselected or the **Reduce all validation errors to warnings** option was changed, a workspace-wide validation occurs.
 - If **Disable validation** is selected, all ADF Validation Markers are deleted.
 - Open page definition editors immediately reflect the changes.

2.2.7 Refactoring ADF Bindings

You can perform several refactoring options on ADF data binding artifacts using the AppXRay dependency engine. For more information, see [Section 2.8.4, "Refactoring ADF Data Binding Artifacts."](#)

2.3 Working with Oracle ADF Controller

In the controller layer, where handling page flow of your web applications is a key concern, ADF Controller provides an enhanced navigation and state management model on top of JSF. OEPE allows you to declaratively create task flows where you can pass application control between different types of activities, such as pages, methods on managed beans, case statements, or calls to other task flows.

These task flows can be reused, and can also be nested, both within themselves and within pages. Task flows nested in pages become regions that contain their own set of navigable pages, allowing users to view a number of different pages and functionality without leaving the main page.

2.3.1 Understanding ADF Task Flows

ADF task flows provide a modular approach for defining control flow in an ADF application. Instead of representing an application as a single large JSF page flow, you can break it up into a collection of reusable task flows. Each task flow contains a portion of the application's navigational graph. The nodes in the task flows are activities. An activity node represents a simple logical operation such as displaying a page, executing application logic, or calling another task flow. The transitions between the activities are called control flow cases.

For more information about ADF Task Flows, see the Introduction to ADF Task Flows section in the "Getting Started with ADF Task Flows" chapter of *Developing Fusion Web Applications with Oracle Application Development Framework*.

2.3.2 Creating a New Task Flow

You can create a task flow.

To create a new task flow:

1. In the Project Explorer, right-click your web project and select **ADF Task Flow** in the context menu. If the ADF Task Flow menu item is not visible, select **Other**, and in the New Gallery, expand **Application Development Framework** and double-click **ADF Task Flow**.
2. In the Create ADF Task Flow wizard, specify a file name (default: `task-flow.xml`) and location (default: `WebContent/WEB-INF` folder of your web project) for the ADF Task Flow. Then click **Next**.
3. On the Task Flow Options page, select options to specify the type of task flow you want to create. The **Create as Bounded Task Flow** checkbox is selected by default. Deselect it to create a source file that will be incorporated into the application's unbounded task flow. Select **Task flow will use page fragments** if you want the view activities that you add to the task flow to reference page fragments files (`.jsff`). Leave it unselected if you want the view activities that you add to the task flow to reference JSF pages. Then click **Finish**. The task flow that you created opens by default in the Diagram view of the ADF Task Flow Editor.
4. After you create the task flow, you can update it using the Diagram, Overview, or Source views. When you use the Design view, the Properties pane is

context-aware, making additional configuration for various activities easily accessible without switching to the Overview pane.

A new XML source file is created every time you create a new ADF unbounded or bounded task flow. By default, the XML source file for an ADF unbounded task flow is called **adfc-config.xml**.

Note: Do not use the same name for more than one task flow in the same workspace, even if the task flows belong to different projects. Identical task flow names can cause unpredictable behavior while debugging.

2.3.3 Adding Activities to a Task Flow

You can add activities to a task flow.

To add activities to a task flow:

Drag an activity from the Palette onto the ADF Task Flow. Normally, you would start with a view activity.

- If you drag a view activity onto the diagram, you can double-click it to display the wizard for the page or page fragment that the task flow is configured to invoke. Use the wizard to define characteristics for the page or page fragment.
- If you drag a router activity onto the diagram, you can use the Properties pane to create an expression whose evaluation will determine which control flow rule will be followed.
- If you drag a method call activity onto the diagram, you can use the Properties pane to configure the method to be called.
- If you drag a task flow call activity onto the diagram, you can double-click it to display the Create Bounded Task Flow dialog where you can define settings for a new bounded task flow.
- If you are creating a bounded task flow, and you drag a task flow return activity onto the diagram, you can use the Properties pane to configure the activity.

Note: The default activity is the first activity to execute in an ADF bounded task flow. For example, the default activity always executes first when a task flow call activity passes control to the ADF bounded task flow. The first activity that you add to a new ADF bounded task flow diagram is automatically identified as the default activity. A green blob over a node indicates that it is the default activity.

For more information on activity types, see the Introduction to Activity Types section in the "Working with Task Flow Activities" chapter of *Developing Fusion Web Applications with Oracle Application Development Framework*.

2.3.4 Adding ADF Bindings to a Task Flow

Once you have created your task flow, you can add ADF bindings to the Router, Method Call, and TaskFlow Call activities by creating Page Definitions. For more information, see [Section 2.3.2, "Creating a New Task Flow."](#)

To create a Page Definition for binding task flow activities:

1. Drop the activity onto the task flow page. ADF bindings can only be added to the Method Call, Router, and TaskFlow Call activities on a task flow.
2. Mouse over the activity to view the Create Page Definition icon. Click on that icon to create a new Page Definition, or to open an existing Page Definition.
3. Once you have a Page Definition for that activity, create a binding. For more information, see [Section 2.2.4.9, "Adding Bindings and Executables."](#)

Figure 2–12 shows an example of a Method Call on a task flow page showing the Create Page Definition option.

Figure 2–12 Method Call on a Task Flow Page

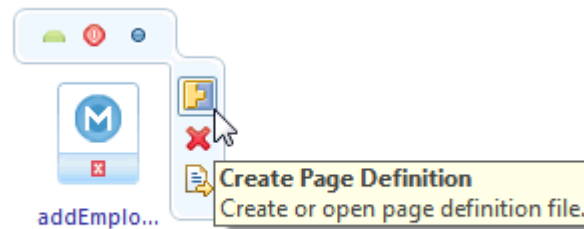


Figure 2–13 shows an example of a Router on a task flow page showing the Create Page Definition option.

Figure 2–13 Router on a Task Flow Page

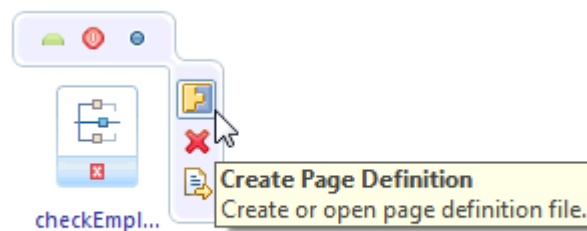
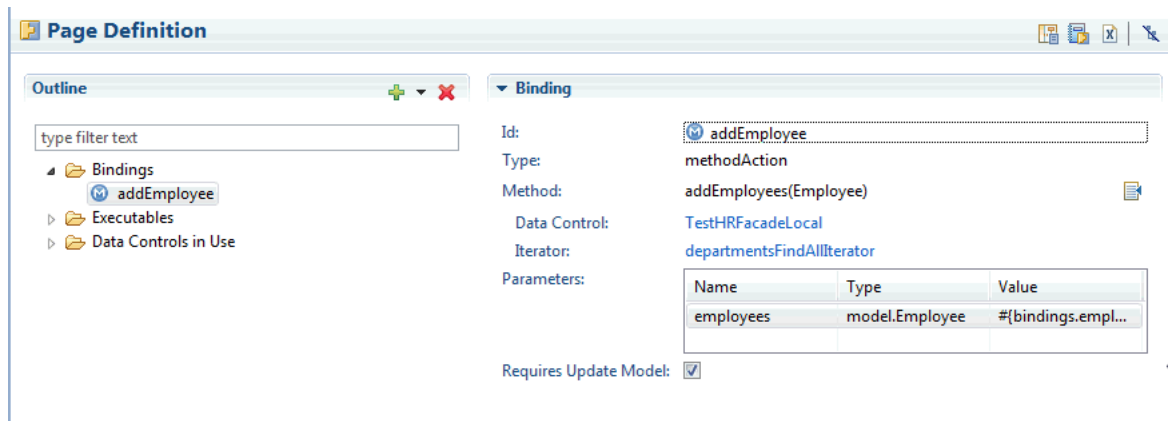


Figure 2–14 shows an example of a `methodAction` binding on the Page Definition that can be bound to a method call activity.

Figure 2–14 *methodAction binding on the Page Definition*

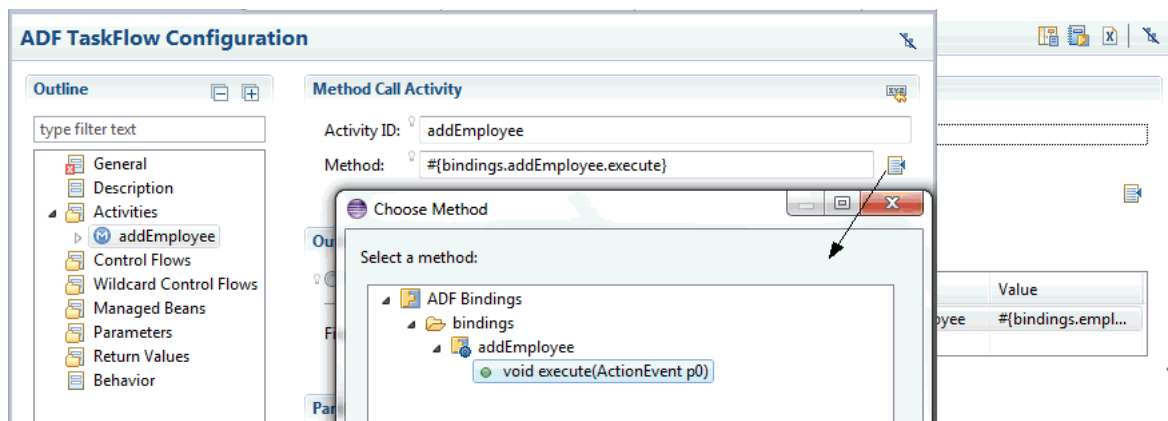
To configure ADF bindings for Method Calls in a task flow:

1. Create a Page Definition for the Method Call, if you don't already have one, and create a binding. For more information, see [Section 2.3.4, "Adding ADF Bindings to a Task Flow."](#)

In the Task Flow editor, choose the **Overview** tab.

2. In the Outline panel on the left, select the activity for which you are going to associate the method call to a binding.
3. In the Method Call Activity panel on the right, select the edit page icon next to the Method field. The Choose Method dialog appears. The bindings available for that method are shown.
4. Choose the appropriate binding and click **OK**.

Figure 2–15 shows an example of the Choose Method dialog for a Method Call taskflow page definition.

Figure 2–15 *Choose Method Dialog*

To configure ADF bindings for Router activities in a task flow:

1. Create a Page Definition for the Router, if you don't already have one, and create a binding. For more information, see [Section 2.3.4, "Adding ADF Bindings to a Task Flow."](#)

In the Task Flow editor, choose the **Overview** tab.

2. In the Outline panel on the left, select the activity for which you are going to associate the Router to a binding.
3. In the Router Activity panel on the right, select the edit page icon next to the Router field. The bindings available for that router are shown.
4. Choose the appropriate binding and click **OK**.

Figure 2–16 shows an example of an attribute binding to be used to bind a Router in a taskflow.

Figure 2–16 Attribute Binding Used to Bind a Router in as Taskflow

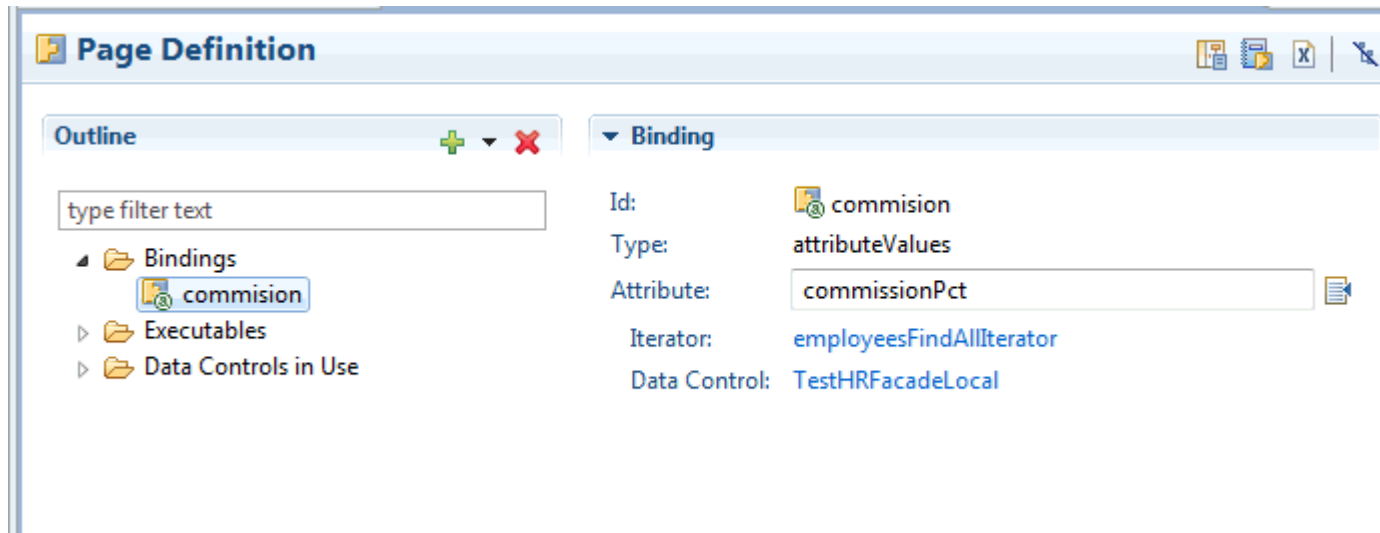
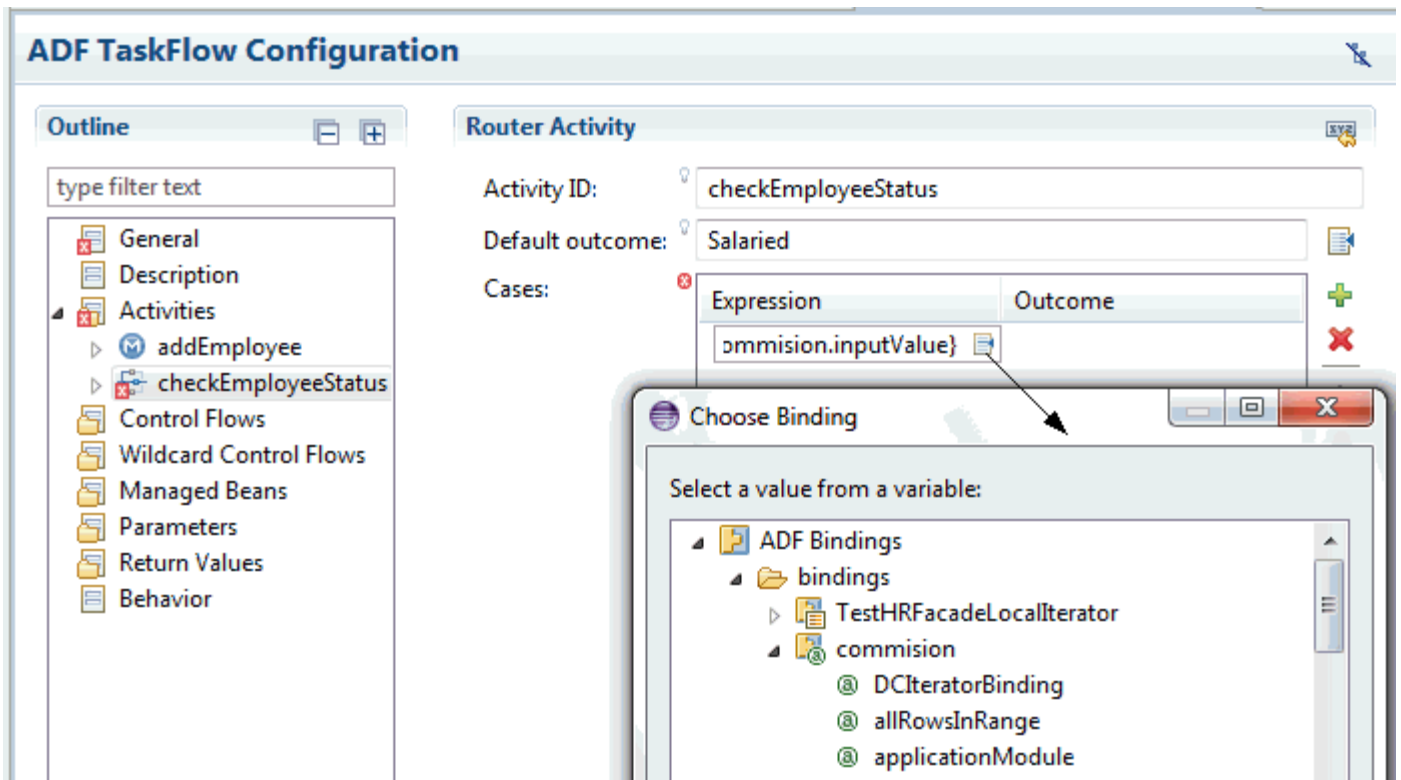


Figure 2–17 shows an example showing the Choose Binding dialog for the Router activity.

Figure 2–17 Choose Binding Dialog



2.3.5 Adding Control Flows to a Task Flow

A control flow case identifies how control passes from one activity to the next in the application.

To add a control flow case:

1. In the Palette, select **Control Flow Case**.
2. On the diagram, click a source activity, for example a view, and then click the destination activity.
3. Set the outcome value in the Properties pane, using either the **From Action** attribute (if the outcome is to be determine by a method) or the **From Outcome** attribute (if the outcome can be set as a String).

2.3.6 Using Task Flows as Regions

You can render a bounded task flow in a JSF page or page fragment (.jsff) by using an ADF region. When first rendered, the ADF region's content is that of the first view activity in the bounded task flow. The view activities used in the bounded task flow must be associated with page fragments, not pages.

You can pass values to the ADF Region using task flow binding input parameters or contextual events.

For more information on ADF Regions, see the Introduction to Using Task Flows in ADF Regions section in the "Using Task Flows as Regions" chapter of *Developing Fusion Web Applications with Oracle Application Development Framework*.

Before you create an ADF region, you need to do the following:

- Create a bounded task flow with one or more view activities associated with page fragments or one task flow call activity to a task flow with view activities.
- Create a page to host the ADF region.

To create an ADF Region:

1. In the Project Explorer, drag the bounded task flow onto the JSF page and drop it where you want to place the ADF region.
2. In the context menu, select **Region**.
3. Review or modify (as appropriate) the following properties which OEPE automatically populates with default values in the Properties pane for the ADF region:
 - **Id**: An ID that the JSF page uses to reference the ADF region.
 - **Rendered**: If selected (the default state), the ADF region renders when the JSF page renders.
 - **Value**: An EL reference to the ADF region model, for example, `{bindings.task_flow1.regionModel}`. This is the region model that describes the behavior of the region.

2.3.7 Running an ADF Task Flow

The procedure for running and debugging task flows differs depending on whether the task flow is bounded or unbounded, whether it contains pages or page fragments.

To run or debug a bounded task flow that uses pages:

- Right-click the bounded task flow in the Project Explorer and choose either Run As or Debug As.

To run or debug a bounded task flow that uses page fragments:

1. Create a JSF page containing a region that is bound to the bounded task flow. When you drop a bounded task flow containing page fragments onto a JSF page, OEPE automatically prompts you to create a region.
2. Create a view activity in the project's unbounded task flow that refers to the page.
3. Right-click the view activity in the Project Explorer and choose **Run**.

For more information on running and debugging task flows, see the Testing ADF Task Flows section in the "Getting Started with ADF Task Flows" chapter of *Developing Fusion Web Applications with Oracle Application Development Framework*.

2.4 Working with Oracle ADF Faces

ADF Faces rich client (known also as ADF Faces) is a set of JavaServer Faces (JSF) components that include built-in Asynchronous JavaScript and XML (AJAX) functionality. While AJAX brings rich client-like functionality to browser-based applications, using JSF provides server-side control, which reduces the amount of JavaScript code that application developers need to write in order to implement AJAX-based applications. In addition to providing a rich set of JSF components, the ADF Faces rich client framework (RCF) provides a client-side programming model familiar to developers accustomed to the JSF development model.

For more information on Oracle ADF Faces, see the "Introduction to ADF Faces Rich Client" chapter of *Developing Fusion Web Applications with Oracle Application Development Framework*.

2.4.1 About ADF Faces Configuration Files

A JSF web application requires a specific set of configuration files, namely, `web.xml` and `faces-config.xml`. ADF applications also store configuration information in the `adf-config.xml` and `adf-settings.xml` files. Because ADF Faces shares the same code base with MyFaces Trinidad, a JSF application that uses ADF Faces components for the UI also must include a `trinidad-config.xml` file, and optionally a `trinidad-skins.xml` file.

For more information on ADF Faces configuration files, see the "ADF Faces Configuration" appendix in *Developing Fusion Web Applications with Oracle Application Development Framework*.

For more information on all Oracle ADF XML files generated by OEPE, see [Section 2.11, "Appendix A Oracle ADF XML Files."](#)

2.4.2 About ADF Data Visualization Components

The ADF Data Visualization components provide significant graphical and tabular capabilities for displaying and analyzing data, and support the use of ADF data controls.

For more information regarding ADF Data Visualization Components, see the "Introduction to ADF Data Visualization Components" chapter of *Developing Fusion Web Applications with Oracle Application Development Framework*.

2.4.3 Working with ADF tags in JSP Pages

You can create JSP pages.

To create a JSP page:

1. In the Project Explorer, In your dynamic web project, right-click the WebContent node, and select **New > JSP File**. Alternatively, from the main menu, click **New > Other**, and choose **JSP File** under the Web node.
2. In the New JSP File dialog, enter or select the parent folder, and enter a file name, for example, `login.jspx`, in the **File name** field. Then click **Next**.
3. On the Select JSP Template page, select a template and view the statements generated for it in the Preview pane. A variety of JSP templates are available for JSP, JSF, and ADF development. For ADF applications, you will need to select a JSP template that supports XML style syntax, for example, **New ADF Rich Faces Page - Basic (xhtml, xml syntax)**.
4. Optionally, you can click the JSP Templates link at the bottom to customize existing templates or create new ones.
5. Click **Next**. The page opens in the Web Page Editor.

2.4.4 Support for ADF Components in the Palette

The Palette pane displays all the available library components. You can click on an item in the palette to expand it.

The ADF Data Visualizations node shows all the GUI components available to represent data, for example, Bars, Pies, or Gauges. You can drag an item from the Palette and drop it on the JSP page. The following figure shows the ADF Data Visualizations node in the Palette.

The last item in the Palette is the Data Palette. Expand it to see content related to the available variables. Variables displayed in the Data Palette range from local Page Variables declared within the current JSP page to JSF Managed Beans available to the whole application. The Data Palette enables easy navigation to variable and class declarations as well as Drag and Drop onto the page.

2.4.5 Using the Tag Drop Editor for ADF Faces Components

You can set properties for ADF components.

To set properties for ADF components using the tag drop editor:

1. Select an ADF Faces component in the Palette, for example, **Form**.
2. Drag the selected component and drop it on to your JSP file.
3. In the dialog that opens, set properties for the ADF component. For example, if you dropped the Form tag, you can specify the type of form to create, as well as select properties that will be used as form fields.

Note: Some tags may not render correctly; rendering support for these will be added in a future release.

2.4.6 Using the Smart Editor for ADF Components

The Properties pane provides a smart editor for ADF components where you can review and update the properties of ADF tags and attributes.

To use the smart editor for an ADF component:

1. Click an ADF component, for example, `af:form`, either in the source view or design view of the JSPX file.
2. Click the **Properties** pane.
3. In the Properties pane, you click any of the property categories, and review or update the attributes. For example, if you are viewing the properties for the `af:form` tag, you can click **Common** and edit the `Id` attribute. If you want to view all attributes together, click **All**.
4. You can click on a hyperlinked field to open its value in the editor.
5. For documentation on a particular tag, press F1 from within the Properties pane.

2.5 Deploying an Oracle ADF Application

You can quickly deploy applications.

To run an ADF application on WLS:

1. Right-click the page you want to run, for example, `login.jspx`, and select **Run As > Run on Server**.
2. In the Run on Server dialog, select **Choose an existing server** if you already have a valid server connection to WLS 10.3.5. If you do not have an existing valid server configuration, select **Manually define a new server** and follow the instructions.

2.6 Debugging an Oracle ADF Application

Like any debugging task, debugging the web application's interaction with Oracle Application Development Framework (Oracle ADF) is a process of isolating specific contributing factors.

To identify and fix application problems, the ADF Debugger provides declarative breakpoints that you can set at the ADF object level (such as task flows and ADF lifecycle phases), as well as standard Java breakpoints. ADF declarative breakpoints provide a high-level object view for debugging ADF applications. For example, you can break before a task flow activity to see what parameters would be passed to the task flow. To perform the same function using only Java breakpoints would require you to know which class or method to place the breakpoint in. ADF declarative breakpoints should be the first choice for ADF applications.

For more information about debugging an ADF application, see "Using the ADF Declarative Debugger" in *Developing Fusion Web Applications with Oracle Application Development Framework*.

2.6.1 Using ADF Source Code with the Debugger

If you have valid Oracle ADF support, you can obtain complete source code for Oracle ADF by opening a service request with Oracle Worldwide Support. You can request a specific version of the Oracle ADF source code. You may be given download and password information to decrypt the source code ZIP file. Contact Oracle Worldwide Support for more information.

Adding Oracle ADF source code access to your application debugging session will:

- Enhance the use of Java code breakpoints by displaying the Oracle source code that's being executed when the breakpoint is encountered. You can also set breakpoints easier by clicking on the margin in the source code line you want to break on. Without the source code, you will have to know the class, method, or line number in order to set a breakpoint within Oracle code.
- For Java code breakpoints set within the source code, you will be able to see the values of all local variables and member fields in the debugger.

After you have received or downloaded the "outer" ZIP, unzip it with the provided password to access the actual source code ZIP file. The ADF source code ZIP name should be a variant of the ADF version number and build number. For example, the ADF source ZIP may have a format similar to `adf_vvvv_nnnn_source.zip`, where `vvvv` is the version number and `nnnn` is the build number. Extract the zip into a folder that has the same name as the zip. For example, if the ADF zip is named `adf_111150_6013_source`, extract it into a folder called `adf_111150_6013_source`.

To add an ADF source library to a project:

1. From the main menu, select **Window > Preferences**.
2. In the Preferences dialog, select **ADF > ADF Source Code Location**.
3. In the ADF Source Code Locations page, click **Add**.
4. In the Add ADF Source Bundle Location dialog, navigate to the ADF source location by clicking **Browse** in the **Location** field. The Bundle information field is populated. Then click **OK**.
5. Click **OK** in the Preferences dialog.

Note: When multiple versions of WLS (for example, 10.3.5 and 10.3.6) and ADF runtime are configured in OEPE, and the shared libraries from these distributions have the same sub-version, only the latest WAR/JAR from the latest WLS runtime will be registered in the OEPE shared library registry. If the version of ADF source code (for example, 11.1.1.6.0) provided to OEPE is targeted to an older WLS/ADF runtime, it will not be mapped to the later version of the runtime JAR.

2.6.2 Setting ADF Declarative Breakpoints

You use the ADF Declarative Debugger features in OEPE to declaratively set breakpoints on ADF task flow activities, ADF bindings, and ADF lifecycle phases. ADF declarative breakpoints provide a high-level object view for debugging ADF applications. For example, you can break before a task flow activity to see what parameters would be passed to the task flow. To perform the same function using only Java breakpoints would require you to know which class or method to place the breakpoint in. ADF declarative breakpoints should be the first choice for ADF applications.

The ADF Declarative Debugger also supports standard Java code breakpoints. You can set Java code breakpoints in any ADF application. You may be able to use Java code breakpoints when an ADF declarative breakpoint does not break in the place you want.

The ADF Declarative Debugger is built on top of the Java debugger, so it has the features and behaviors of the Java debugger. But instead of needing to know the Java class or method, you can set ADF declarative breakpoints in visual editors.

2.6.3 Setting and Using ADF Task Flow Breakpoints

You can add breakpoints to task flow activities in the task flow editor by selecting a task flow activity and using the context menu to toggle or disable breakpoints on that activity. After the application pauses at the breakpoint, you can view the runtime structure of the objects as well as a list of data for a selected object in the ADF Structure and Data window.

When an ADF declarative breakpoint is set, it appears as a red dot icon in the task flow activity.

To set and use a breakpoint on a task flow activity:

1. Double-click the task flow in the Project Explorer to open in the task flow editor.
2. Right-click a task flow activity and choose **Toggle Breakpoint** from the context menu.

A blue dot appears on the task flow activity, signifying that the breakpoint has been set.

3. Start the debugging process. You can:
 - From the main menu, choose **Run > Debug As**, and then select the server.
 - From the Project Explorer, right-click the project, `adfc-config.xml`, `faces-config.xml`, task flow, or page and choose **Debug As**, and then select the server.

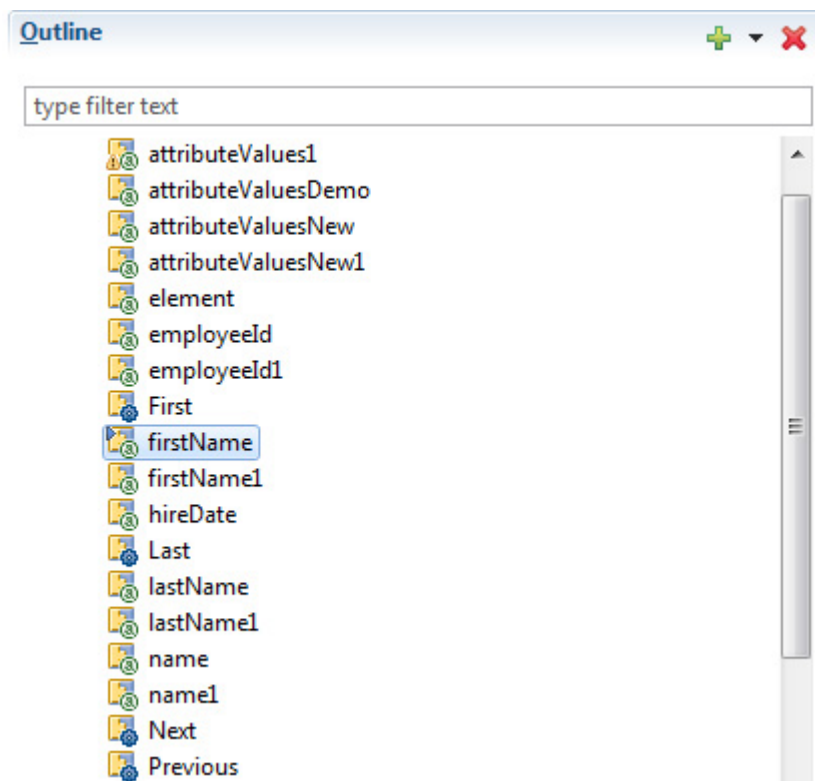
4. When the application is paused at a breakpoint, a red triangle appears next to the breakpoint icon on the task flow activity. You can now examine the application using the ADF Structure and Data window or the Debug window.
5. The ADF Structure and Data window appears by default. You can use this window to examine runtime structure and corresponding data values.
6. Select a node in the ADF structure in the left pane and view pertinent data in the right pane. Task flow activity declarative breakpoints pause the application just before the task flow activity is executed. You can use the **Step Into** (F5) function to pause the application just prior to executing the called task flow default activity.
7. Continue debugging the application as required.

2.6.4 Setting and Using ADF Page Definition Breakpoints

You can add breakpoints to page definition executables and bindings in the page definition editor by selecting a binding or executable item and using the context menu to toggle or disable 'before' or 'after' breakpoints on that item. After the application pauses at the breakpoint, you can view the runtime structure of the objects as well as a list of data for a selected object in the ADF Structure and Data window.

To set and use ADF page definition breakpoints:

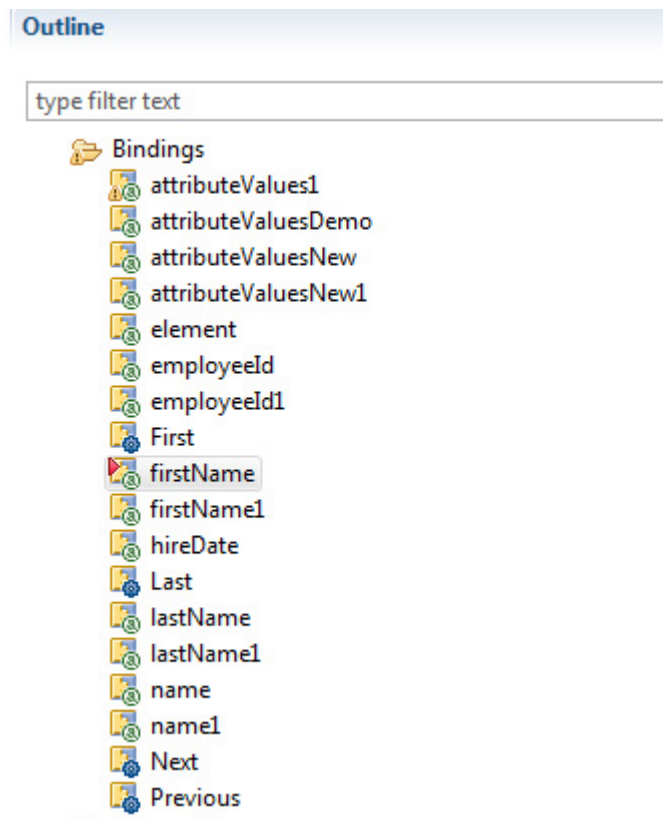
1. In the page definition editor, in the Outline section, right-click the binding or executable you want to set the breakpoint on. From the context menu, select **Toggle Breakpoint Before** or **Toggle Breakpoint After** or both depending on where you want to place the breakpoint. A blue triangle appears on the item indicating that the breakpoint has been set.



2. From the main menu, select **Window > Open Perspective > Debug**. The Breakpoints window lists the breakpoints you have set selected by default. You can deselect breakpoints in the Breakpoints window if you do not want them to be

considered by the debugger. Double-clicking a breakpoint in the Breakpoints window opens the page definition editor with the particular binding or executable in focus.

3. Start the debugging process. You can:
 - From the main menu, choose **Run > Debug As**, and then select the server.
 - From the Project Explorer, right-click the project or page and choose **Debug As**, and then select the server.
4. When the application is paused at an ADF page definition breakpoint, a red triangle appears on the impacted binding element in the page definition editor, as shown in the following figure. You can now examine the application using the debugging resources available.



5. The ADF Structure and Data window appears by default when you work in the **Debug** perspective. You can use this window to examine runtime structure and corresponding data values.
6. Select a node in the ADF structure and view pertinent data in the right pane.
7. Continue debugging the application as required.

2.6.5 Setting and Using ADF Lifecycle Phase Breakpoints

You can set Before and After breakpoints on all the ADF lifecycle phases in the ADF Lifecycle diagram. For each phase, you can set Before only, After only, or both. You can set breakpoints on as many phases as you want. The ADF Lifecycle diagram is available from the Breakpoints window. You can set ADF lifecycle breakpoints on any of the ADF lifecycle phases:

- JSF Restore View
- Initialize Content
- Prepare Model
- JSF Apply Request Values
- Apply Input Values
- JSF Process Validations
- Validate Input Values
- Process Update Model
- JSF Update Model Values
- Validate Model Updates
- JSF Invoke Application
- Process Component Events
- Metadata Commit
- Prepare Render
- JSF Render Response

To set and use ADF lifecycle phase breakpoints:

1. Select **Window > Show View > Breakpoints** to open the Breakpoints window.
2. In the Breakpoints window, click the **Add ADF Lifecycle Breakpoints** button in the toolbar.

The ADF Lifecycle Diagram is displayed.

3. In the ADF Lifecycle Diagram, select the left checkbox to set a breakpoint before the phase, or select the right checkbox to set a breakpoint after the phase, or select both.
4. Start the debugging process. You can:
 - From the main menu, choose **Run > Debug As**, and then select the server.
 - From the Project Explorer, right-click the project, `adfc-config.xml`, `faces-config.xml`, task flow, or page and choose **Debug As**, and then select the server.
5. When the application is paused at an ADF lifecycle phase breakpoint, you can examine the application using the debugging resources available.
6. The ADF Structure and Data window appears by default when you work in the Debug perspective. You can use this window to examine runtime structure and corresponding data values.
7. Select a node in the ADF structure and view pertinent data in the right pane.
8. Continue debugging the application as required.

2.6.6 Using the EL Expression Evaluator

When the application is paused at a breakpoint, you can use the EL expression evaluator to enter an EL expression for evaluation. You can enter arbitrary EL expressions for evaluation within the current context. If the EL expression no longer

applies within the current context, the value will be evaluated to null. The EL Evaluator is available for debugging any JSF application.

To use the EL Evaluator:

1. Set a break point in the JSF application.
2. Start the debugging process.
3. When the breakpoint is reached, click the **ADF EL Evaluator** tab to bring it forward.
4. Enter an EL expression in the **Expression** field.

When you click in the field after entering #{ or after a period, a discovery function provides a selectable list of expression items. Auto-completion will be provided for easy entry. You can evaluate several EL expressions at the same time by separating them with semicolons.

5. When you finish entering the EL expression, click Evaluate and the expression is evaluated.

2.6.7 Using the ADF Structure and Data Window

The ADF Structure and Data window displays the ADF structure on the left pane, and relevant data for a given object in the right pane.

When you use the Debug perspective of OEPE, the ADF Structure and Data window opens by default. To manually launch the ADF Structure and Data window, choose Window from the main menu, and then select **Show View > ADF Structure and Data**.

2.6.7.1 Using the ADF Structure Pane

When the application is paused at a breakpoint, the ADF Structure pane displays a tree structure of the ADF runtime objects and their relationships within the application. In particular, it shows the hierarchy of view ports, which represent either the main browser window or contained regions. When you select different items in the ADF Structure pane, the data display in the accompanying ADF Data pane changes.

The roots of the hierarchy are the sibling nodes Scopes and ADF Context:

- **Scopes:** Displayed at the top of the ADF Structure hierarchy above its sibling ADF Context node. There is only one Scopes node in the ADF Structure hierarchy. You can expand the Scopes node to show a list of child scope nodes (such as viewScope and pageFlowScope). If you select a child scope node, the ADF Data pane displays the variables and values for that scope.
- **ADF Context:** Displayed as the root node of the ADF Structure hierarchy below its sibling Scopes node. There will only be one ADF Context within the ADF Structure hierarchy.

2.6.7.2 Using the ADF Data Pane

When an application is paused at an ADF declarative breakpoint, the ADF Data pane (in the ADF Structure and Data window) displays relevant data based on the selection in the ADF Structure pane.

The ADF Data pane displays the following types of data:

- You can inspect the values of requestScope, viewScope, pageFlowScope, applicationScope, and sessionScope by expanding each corresponding node in the ADF Structure pane.

- When the ADF context is selected in the ADF Structure pane, the current value of the ADF context variables will be displayed in the ADF Data pane.
- Selecting a view port within the ADF Structure hierarchy will display the view port's current view port details in the ADF Data pane.
- In the ADF Structure pane, each individual ADF task flow within a page flow stack hierarchy is selectable. An ADF task flow selected in the ADF Structure pane will display the current task flow information in the ADF Data pane.
- When you select a page or page fragment node in the ADF Structure hierarchy, the corresponding UI component tree is displayed within the ADF Data pane.

2.7 Using AppXray for Oracle ADF Artifacts

AppXRay is a central feature of Oracle Enterprise Pack for Eclipse designed for dependency tracking, validation, visualization, and refactoring support. In this release of OEPE, AppXRay is enabled for ADF components as well.

Some of the features in the source view of the Web Page editor that are driven by AppXray are:

- Hovering over a component displays the properties of the component. For example, if you hover over a managed bean in source editor, its properties (**Name**, **Type**, and **Scope**) are displayed.
- Hovering over a component and pressing Ctrl + Space displays a popup with possible code values.
- Hovering over a component and pressing Ctrl results in a hyperlink that leads you to the tag documentation.
- Content assist for ADF bindings on a page as defined by the page definition file.

To use AppXaminer to view dependency relationships:

1. In the Project Explorer, right-click any file in your application, for example, `login.jspx`, and select **Show AppXray Dependencies** from the context menu.
2. AppXaminer opens in the Editor displaying the relationship the selected page has with other components. Numeric values indicate the number of references a component has with another.
3. Expand a node to the relationship it has with other components.
4. Right-click a node and select **Show Reference Detail** from context, which invokes a popup window displaying the detailed components involved. Alternatively, select **Open** from the context menu to view the file in the editor.

2.8 Refactoring Oracle ADF Components

OEPE provides refactoring options to rename, move, and delete the ADF components that your application uses. These refactoring options synchronize your changes with other parts of the application that are dependent on the changes.

2.8.1 Refactoring ADF Pages

This section describes the refactoring options available for ADF pages. [Table 2-1](#) lists each refactoring operation and what components are consequently modified.

Table 2–1 Refactoring Options Available for ADF Pages

Refactoring Operation	Components Modified
Renaming a page file	<ul style="list-style-type: none"> ■ faces-config.xml ■ ADF task flow ■ databindings.cpx
Moving a page to a different folder	<ul style="list-style-type: none"> ■ faces-config.xml ■ ADF task flow ■ databindings.cpx
Deleting a page file	<p>The corresponding page definition file and the entry for the page definition in the CPX file are deleted</p> <p>Note: If there are invalid references, an error message is displayed</p>
Deleting a folder containing a page	The corresponding page definition files and the entries for the page definitions in the CPX file are deleted
Renaming a folder containing a page	<ul style="list-style-type: none"> ■ faces-config.xml ■ ADF task flow ■ corresponding page definition file ■ databindings.cpx

2.8.2 Refactoring ADF Task Flow configuration files

[Table 2–2](#) describes the refactoring options available for ADF task flow configuration files. The following content lists each refactoring operation and what components are consequently modified.

Table 2–2 Refactoring Options for ADF Task Flow Configuration Files

Refactoring Operation	Components Modified
Renaming the task-flow-id tag of a bounded task flow	<ul style="list-style-type: none"> ■ task-flow-call tag in the task flow configuration file ■ Page definition XML file of the page in which the task flow is embedded
Renaming the task flow configuration file	<ul style="list-style-type: none"> ■ document tag in the task flow configuration file ■ Page definition XML file of the page in which the task flow is embedded
Renaming the task-flow-activity id attribute for view and taskflow call activities	References to task-flow-activity within the configuration file
Renaming task flow id	ID reference in page definition
Renaming a folder containing a task flow	<ul style="list-style-type: none"> ■ document tag in task flow configuration file ■ Page definition XML file of the page in which the task flow is embedded
Moving a task flow	<ul style="list-style-type: none"> ■ document tag in the task flow configuration file ■ Page definition XML file of the page in which the task flow is embedded
Deleting a task flow	Entry in page definition file deleted

2.8.3 Refactoring JSF/ADF Managed Beans

OEPE support for refactoring a Java class includes all changes except those noted in [Table 2-3](#), which lists each refactoring operation on managed beans and components consequently modified.

Table 2-3 Refactoring Operations on Managed Beans

Refactoring operation on Managed Bean	Components Modified
Moving to a different package	Class name in the managed bean definition in the corresponding configuration file
Deleting the Java class	<ul style="list-style-type: none"> ■ No change to the definition of the bean in the configuration file ■ An error message warns you about invalid references
Renaming the name of the managed bean in the configuration file	References to the managed bean in all pages
Deleting the name of the managed bean or the managed bean itself	Reference to the managed bean in all pages
Renaming a property of the managed bean	<ul style="list-style-type: none"> ■ Method or field in the managed bean class that corresponds to the property ■ Name of property in <code>faces-config.xml</code> ■ EL expression in JSP files
Renaming a Java class that is the type for a managed property in a JSF/ADF managed bean	<ul style="list-style-type: none"> ■ Accessors on the property ■ Reference to the managed bean in all pages

2.8.4 Refactoring ADF Data Binding Artifacts

This section describes the refactoring options available for ADF data binding artifacts. [Table 2-4](#) lists each refactoring operation and what components are consequently modified.

-
- Notes:**
- All ADF Model files (`.dcx`, `.cpx`, page definitions) must reside in the `adfmsrc` folder. The user can move the files to any package under `adfmsrc`. To create a package, right-click the folder and select **New > Package**.
 - Refactoring of Java files, for example, JPA entities and session beans, does not change the corresponding entries in ADF Model files.
-

Table 2-4 Refactoring Operations and Modified Components

Refactoring Operation	Components Modified
Moving <code>datacontrols.dcx</code> to another package under the <code>adfmsrc</code> folder	<ul style="list-style-type: none"> ■ <code>datacontrolusages</code> attribute in the <code>DataBindings.cpx</code> file ■ Reference to the <code>.dcx</code> file in <code>META-INF/adfm.xml</code> of the Model project ■ <code>package</code> attribute in the file

Table 2–4 (Cont.) Refactoring Operations and Modified Components

Refactoring Operation	Components Modified
Renaming or moving a folder containing a data control file	<ul style="list-style-type: none"> ■ datacontrolusages attribute in the DataBindings.cpx file ■ Reference to the .dcx file in META-INF/adfm.xml of the Model project ■ package attribute in the file
Changing the Id within a data control	<ul style="list-style-type: none"> ■ path attribute in the data control usages section of the .cpx file
Moving DataBindings.cpx	<ul style="list-style-type: none"> ■ Reference to the .dcx file in META-INF/adfm.xml of the Model project ■ package attribute in the file
Renaming or moving a folder containing an ADFm file (*.cpx)	<ul style="list-style-type: none"> ■ Reference to the .dcx file in META-INF/adfm.xml of the Model project ■ package attribute in the file
Changing the data control ID in the data control usages section of an ADFm file (*.cpx)	<ul style="list-style-type: none"> ■ DataControl attribute in associated page definition files
Renaming or moving a page definition file	<ul style="list-style-type: none"> ■ Id attribute in the root element ■ Reference to the page definition file in the .cpx file ■ package attribute in the file ■ Page definition Id
Renaming or moving a folder containing a page definition file	<ul style="list-style-type: none"> ■ Reference to the page definition file in the .cpx file ■ package attribute in the file ■ Page definition Id

2.8.5 Externalizing Strings

The string externalization feature in OEPE enables you to extract strings from JSF pages and externalize them in resource bundles. The strings are then substituted by the corresponding EL expression. By externalizing strings, the text can be translated in different languages.

To externalize strings:

1. In the Project Explorer, right-click the page you want to extract strings from. In the context menu, choose **Source > Externalize Strings**.
2. In the Externalize Strings dialog, select the strings you want to externalize. The Externalize Strings dialog contains the following options:
 - **Enter common prefix for generated keys:** Specifies an optional prefix for all newly generated key. A good practice is to use the name of the JSF page to ensure that entries in the property files can be easily grouped.
 - **Strings to externalize:** Displays the list of all strings in the file.
 - **Externalize:** Marks the selected strings to be externalized. Externalized strings will be placed in a property file. In the code, the string is substituted by its corresponding EL expression.
 - **Ignore:** Marks the selected strings to be ignored from externalization.
 - **Edit:** Opens a dialog to enter a new value and key.

- **Context:** Displays the occurrence of the string in the context of the page.
 - **Bundle Property File:** Select a property bundle file. If a properties file has not been created previously, OEPE creates called messages.properties.
3. Click **Next**.
 4. In the next page of the wizard, confirm the changes to be performed after reviewing the original source and the refactored source.
 5. Click **Finish**.

2.8.6 Adding and Refactoring ADF Tag IDs

If you have tags on your page with missing IDs, you can add and refactor your ADF tags using the Fix ADF Component IDs feature from your page editor.

To add and refactor ADF tag IDs:

1. With the focus on your page, right-click and choose **Source > Fix ADF Component IDs**.
2. A refactoring diff window comes up and shows you the code before and after the refactor operation. Inspect the changes to ensure correctness, then click **Finish**.

2.9 Reusing Oracle ADF Application Components

OEPE and ADF enable you to package certain ADF components into the ADF Library for reuse in applications. Reusable ADF components can be task flows and page templates.

2.9.1 About ADF Library Support

ADF Library provides a convenient and practical way to create, deploy, and reuse high-level components. When you first design your application, you design it with component reusability in mind. If you created components that can be reused, you can package them into JAR files and add them to a reusable component repository. If you need a component, you may look into the repository for those components and then add them into your project or application.

An ADF Library JAR contains ADF components and does not and cannot contain other JARs. It should not be confused with OEPE library, Java EE library or Oracle WebLogic shared library.

An ADF Library created in OEPE can be consumed by Oracle JDeveloper.

Note: OEPE does not support generating an ADF library from a project that contains a reference to an external JAR file. In Oracle JDeveloper, the file `adflibREADME.txt` contains references to external JAR files. The ADF Library feature in OEPE does not create the `adflibREADME.txt` file.

2.9.1.1 Naming Conventions

When you create reusable components, you should try to create unique and relevant names for the application, project, task flow, connection, or any other file or component. Do not accept the OEPE wizard default names such as `task-flow-definition.xml`. You want to try to have unique names to avoid naming conflicts with other projects, components, or connections in the application. Naming

conflicts could arise from components created in the consuming application and those loaded from other JAR files.

For more information on best practices for naming reusable ADF components, see "Naming Conventions" in *Developing Fusion Web Applications with Oracle Application Development Framework*.

2.9.2 Creating an ADF Library

You can create ADF libraries.

To export ADF components to an ADF Library:

1. In the Project Explorer, right-click the Dynamic Web project and choose Export > Export from the context menu.
2. On the Select page, choose **Oracle > ADF Library**.
3. On the Export ADF Library page, specify the **Project** (if different from the current one), and the ADF Library you want to export to.
4. In the **Dependent Projects** field, select any dependent projects you want to add to the ADF Library.
5. Click **Finish**.

2.10 Configuring and Using ADF with GlassFish Server

You can configure GlassFish Server to run your Oracle ADF applications, and configure an ADF-enabled GlassFish Server run-time in your Java applications.

For a list of the supported Oracle ADF features for GlassFish, go to the OTN site at <http://www.oracle.com/technetwork/developer-tools/adf/overview/adfessentials-1719844.html>.

The following sections describe how to configure GlassFish Server for use with Oracle ADF:

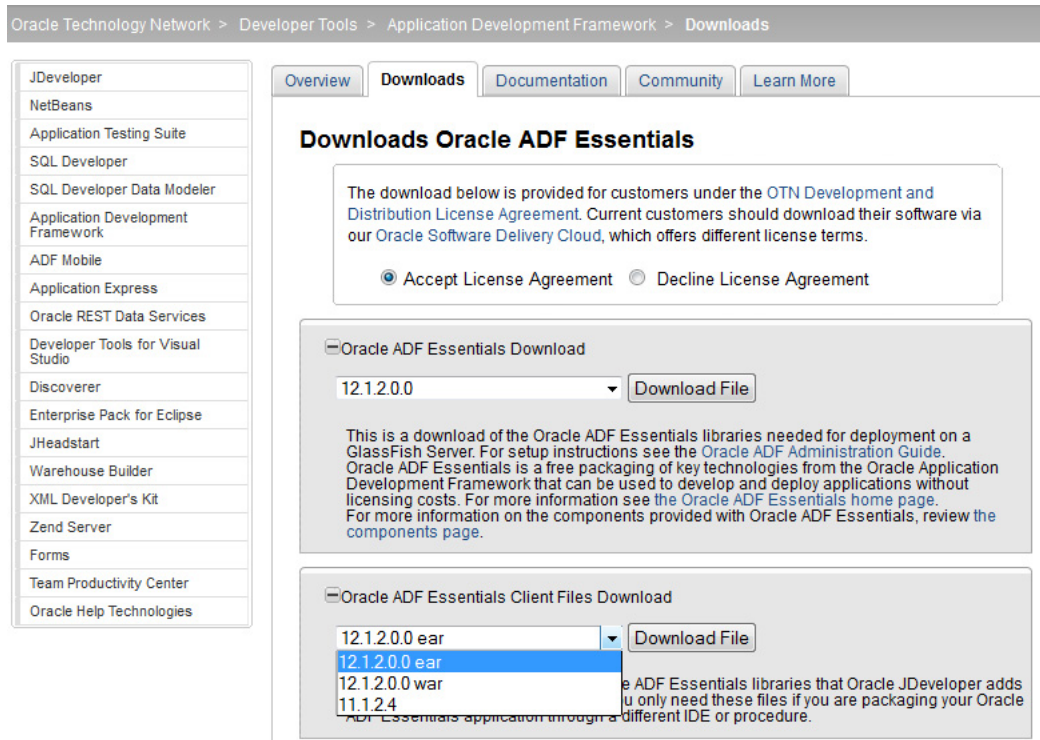
- [Section 2.10.1, "How to Download ADF Essentials."](#)
- [Section 2.10.2, "How to Download and Install GlassFish Server."](#)
- [Section 2.10.3, "How to Configure GlassFish for OEPE"](#)
- [Section 2.10.4, "How to Configure GlassFish for ADF Essentials"](#)
- [Section 2.10.5, "How to Register the ADF Essentials Client WAR Library in Your Workspace"](#)
- [Section 2.10.6, "How to Create an ADF Application that Uses GlassFish Runtime"](#)
- [Section 2.10.7, "How to Create a Global JDBC Data Source"](#)
- [Section 2.10.8, "Known Problems and Solutions"](#)

For more information, see "Configuring GlassFish Server" in *Administering Oracle ADF Applications*.

2.10.1 How to Download ADF Essentials

In order for a GlassFish Server to run Oracle ADF applications, you must download the Oracle ADF Essentials files from the Oracle Technology Network at <http://www.oracle.com/technetwork/developer-tools/adf/downloads/adf-download-1649592.html>, as shown in [Figure 2-18](#).

Figure 2–18 Oracle ADF Essentials Download Page



Download the Oracle ADF Essentials file `adf-essentials.zip` to a temporary location. You will install the ADF Runtime library files into the GlassFish installation directory, as described in [Section 2.10.3, "How to Configure GlassFish for OEPE."](#)

Download the Oracle ADF Essentials Client files `adf-essentials-client-ear.zip` and `adf-essentials-client-war.zip` to a temporary location and then extract the files to flat-structured temporary directories, a separate directory for each download.

For instance, if you are using `unzip`, you can add the `-j` option to create a flat directory structure that has no hierarchical folders.

```
unzip -j <file> -d <destination>
```

From `adf-essentials-client-war.zip`, you will create a user library which you will use when you create an ADF Application, described in [Section 2.10.6, "How to Create an ADF Application that Uses GlassFish Runtime."](#)

Once you have created the ADF Application, you will import the files from `adf-essentials-client-ear.zip` to the `EarContent/lib` folder of your application, described in [Section 2.10.6, "How to Create an ADF Application that Uses GlassFish Runtime."](#)

2.10.2 How to Download and Install GlassFish Server

For instructions on obtaining and installing GlassFish, see <http://glassfish.java.net/downloads/3.1.2-final.html>.

If you install from Zip, after you have unzipped the files you can start the server using `<glassfish_install>/bin/asadmin start-domain`. Alternatively, you can start and stop the server from the Servers pane in OEPE once the server has been defined in OEPE.

If you use the native installer, you can choose from Typical Installation or Custom Installation. Custom Installation allows you to specify a domain name other than `domain1`, and to choose to use different port numbers for the administration and HTTP listener ports. At the end of installation, the server is started.

You can create a GlassFish domain by using the GlassFish Administration utility `asadmin`. For more information see:

- "Using the `asadmin` Utility" in *Oracle GlassFish Server Reference Manual*.
- "create-domain" in *Oracle GlassFish Server Reference Manual*

2.10.3 How to Configure GlassFish for OEPE

You need to create a server configuration for GlassFish Server. It will be the link to the GlassFish Server that you use to run your project.

To configure GlassFish:

1. From the main menu, select **Window > Show View > Servers** to open the Servers pane.
2. In the Servers pane, right-click and select **New > Server**.

Alternatively if there are no servers defined, click the `No servers are available. Click this link to create new server.` link.

3. On the Define a New Server page of the New Server wizard, expand **GlassFish** and select the type of GlassFish server you are running, for example, **GlassFish 3.1.2**. Enter the server's hostname, for example `localhost` and click **Next**.
4. On the server details page of the Define a New Server wizard the default domain directory is displayed.

If necessary, click **Browse** next to **Domain Directory** to open the Browse for Folder dialog. Navigate to the domain location which is typically at `<glassfish_install>/glassfish/domains` and choose the domain. Click **OK**.

If necessary, enter the GlassFish Server Administrator Password, and click **Finish**.

5. The new server connection appears in the Servers pane. You can start and stop the server from the right-mouse menu of the server node.

2.10.4 How to Configure GlassFish for ADF Essentials

Once you have create a server configuration for GlassFish server you can configure the GlassFish domain for ADF Essentials. See:

- [Section 2.10.4.1, "Installing ADF Essentials on a Domain"](#)
- [Section 2.10.4.2, "Installing ADF Essentials on a Domain With a Password"](#)

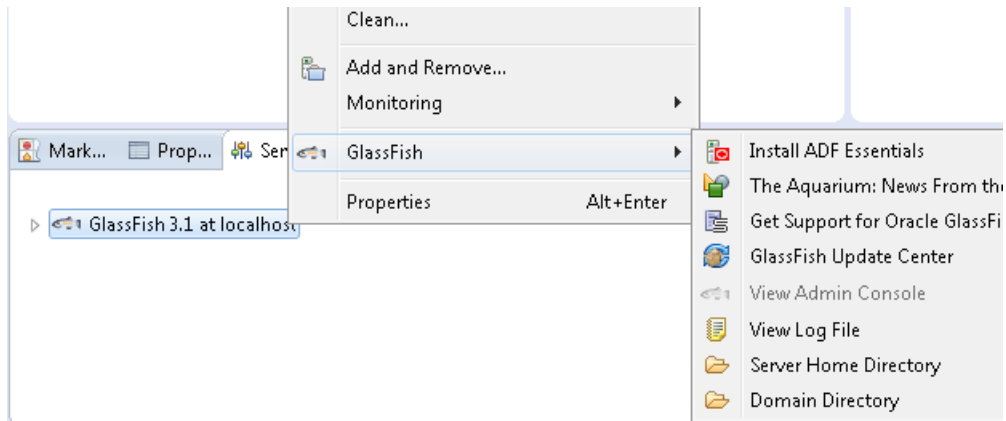
2.10.4.1 Installing ADF Essentials on a Domain

If you are configuring a domain that does not use a password, you can use the server adapter from within OEPE.

To configure a domain that does not use a password:

1. In OEPE, in the Servers pane, right-click the GlassFish server node and choose **GlassFish > Install ADF Essentials**, as shown in [Figure 2-19](#).

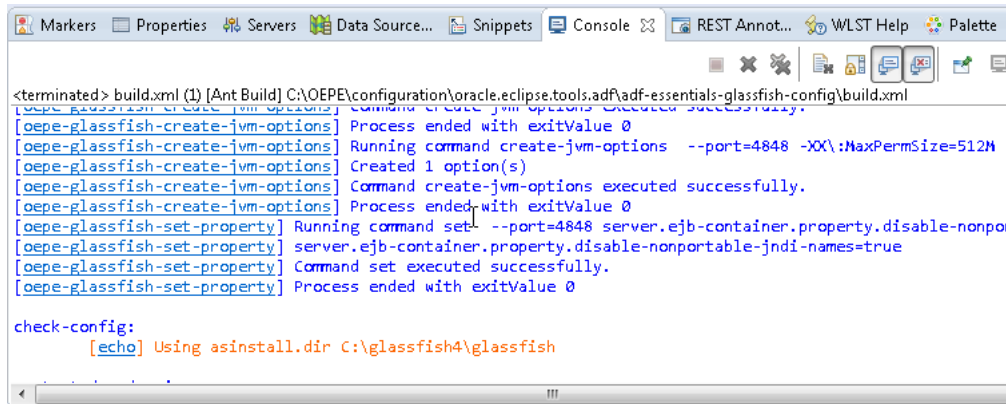
Figure 2–19 GlassFish Menu in Servers Pane



2. Navigate to the `adf-essentials.zip` file that you downloaded from OTN and click **Open**.

You can follow the installation in the console, **Window > Show View > Console**, as shown in [Figure 2–20](#).

Figure 2–20 Importing ADF Essentials Console



Check that the GlassFish server is not stopped in the server pane. This confirms that ADF Essentials is configured correctly and running in the server instance. You now have a connection to a GlassFish Server, which is configured for ADF Essentials.

2.10.4.2 Installing ADF Essentials on a Domain With a Password

If you are installing ADF Essentials on a domain that has been configured to require a password, you must manually configure the domain from the GlassFish console.

To configure a domain that does use a password:

1. Login to the GlassFish admin console, which is at `http://<machine_name>:4848/`. For example, `http://localhost:4848/` when the server runs on the local machine.
2. Navigate to **Configurations > server-config > JVM Settings**.
3. Select **JVM Options** and add the following JVM options:
 - `-Doracle.mds.cache=simple`
 - `XX:MaxPermSize=512m`

4. Once you have set the JVM parameters, restart the server.

Alternatively, you can set these as <jvm-options> entries in the domain.xml file for the domain, <glassfish_install>\glassfish\domains\

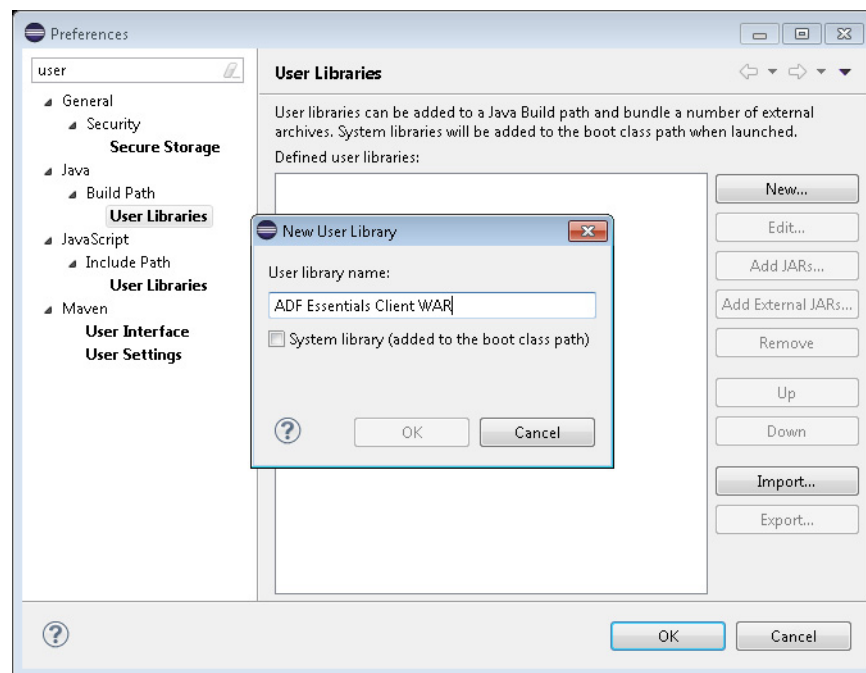
2.10.5 How to Register the ADF Essentials Client WAR Library in Your Workspace

In this procedure, you will add the ADF Essentials Client WAR files to a user library in OEPE so that when you create an ADF Application you can add the user library.

To register your ADF essentials client libraries

1. In OEPE, go to **Windows > Preferences**. Search for "User", and select **Java > Build Path > User Libraries** to display the User Libraries page of the Preferences dialog, as shown in [Figure 2-21](#).

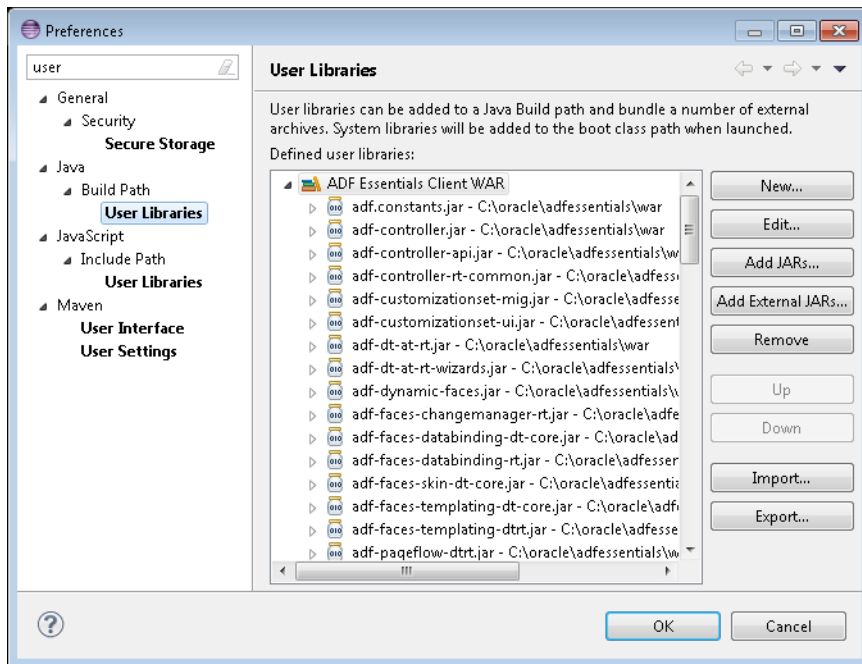
Figure 2-21 User Libraries - Java



2. Click **New** to open the New User Library dialog. Enter a name for the new ADF Essentials Client WAR library and click **OK**.
3. In the User Libraries page of the Preferences dialog, select the new library and click **Add External JARS**.

Navigate to the location where you flat unzipped the adf-essentials-client-war.zip. Select all the files and click **Open**.

The User Libraries page now shows the as shown in [Figure 2-22](#).

Figure 2–22 User Libraries - Java - Add External JARS

4. Click **OK**.

2.10.6 How to Create an ADF Application that Uses GlassFish Runtime

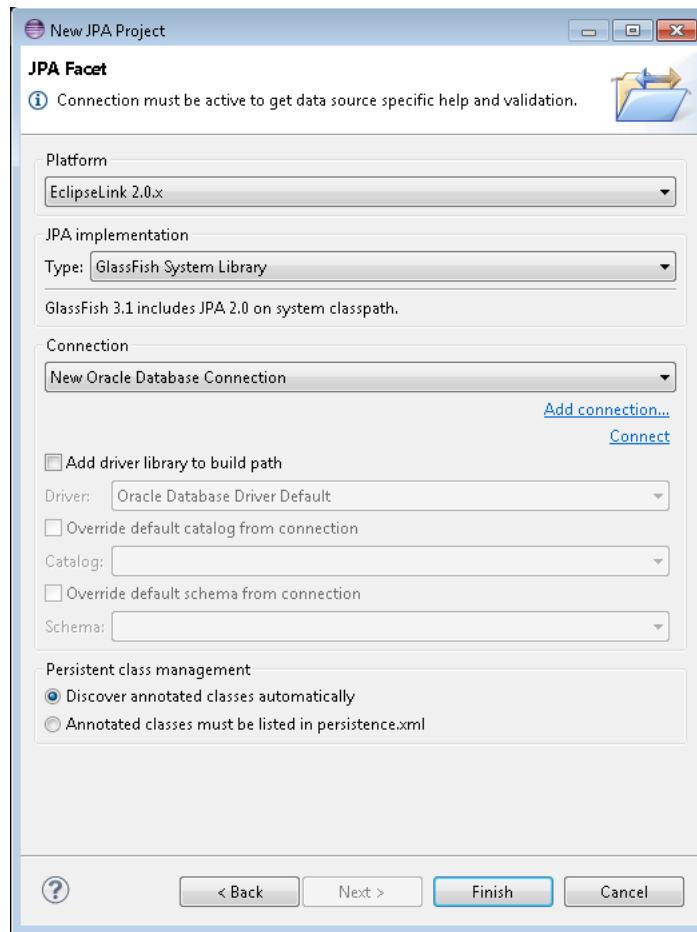
Now that you have installed GlassFish Server, configured a domain for ADF Essentials, and added the ADF Essentials Client WAR library to your workspace, you can create an ADF application.

To create a new ADF Application that uses GlassFish

1. In OEPE, go to **File > New > ADF Application**. The New Oracle ADF Application wizard appears.
2. If you performed the previous procedures correctly, you will see that your GlassFish server is listed in **Target runtime with ADF extension**. If your GlassFish server is not listed, cancel the wizard, and go back and perform the procedures in [Section 2.10.3, "How to Configure GlassFish for OEPE"](#) and [Section 2.10.4, "How to Configure GlassFish for ADF Essentials."](#)

Enter a name for your application.

3. Click **New JPA Project** to open the New JPA Project dialog. You use the JPA project for access to the data for your application.
4. Click **Next** twice to display the JPA Facet page of the New JPA Project dialog, as shown in [Figure 2–23](#).

Figure 2–23 Defining the JPA Facet


If necessary, change the **Platform**. EclipseLink is the reference implementation of JPA, and if you choose it you can take advantage of the EclipseLink shared libraries. For more information, see the "EclipseLink User's Guide - Developing JPA Projects" which is available on the Eclipse site at http://wiki.eclipse.org/EclipseLink/UserGuide/Developing_JPA_Projects_%28ELUG%29.

5. Choose from the list to use an existing database connection, or click **Add connection** to create a new database connection.

Click **Finish** to return to the New Oracle ADF Application wizard.

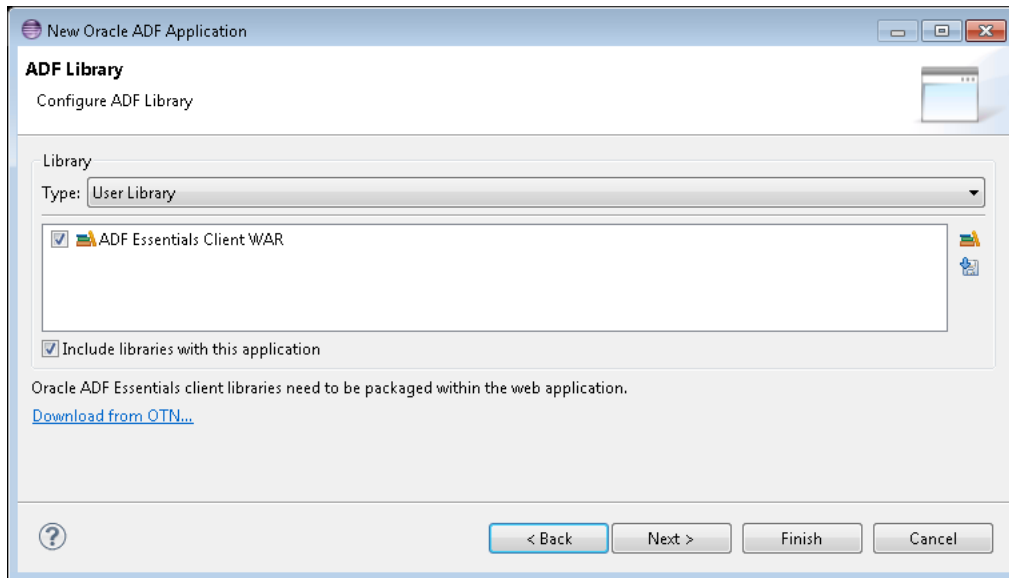
6. In the New Oracle ADF Application dialog, click **Next** to open the ADF Library page. You should be able to see the library that you defined in [Section 2.10.5, "How to Register the ADF Essentials Client WAR Library in Your Workspace"](#).

If you have not yet downloaded the ADF Essentials Client files from OTN, you can download them by clicking **Download from OTN**. Flat unzip the client Zip files to a local location.

If you have downloaded the ADF Essentials Client files from OTN, but not already created the user library for the War client files, click  (Manage libraries) to open the User Libraries dialog, where you can create a user library (under Java > Build Path > User Libraries), to contain the flat unzipped content of `adf-essentials-client-war.zip`.

If the library is not already selected, select it as shown in [Figure 2–24](#), and click **Finish**.

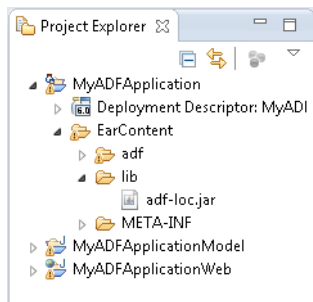
Figure 2–24 ADF Library Page



7. In the New Oracle ADF Application, click **Finish**. The application is created, and the generated projects are displayed in the Project Explorer.
8. The final step is to add the files contained in `adf-essentials-client-ear.zip` to your new project.

In the Project Explorer, expand the `<project_name>` and `EarContent` nodes, as shown in [Figure 2–25](#).

Figure 2–25 ADF Project EAR Library Directory



Right-click `lib` and choose **Import**.

9. In the Import wizard, choose **General > File System** and click **Next**.

In the File System page of the Import wizard, browse to the location where you flat unzipped `adf-essentials-client-ear.zip`.

Select all files and click **Finish**. If you see a dialog asking whether to overwrite a file, select **Yes to All**.

2.10.7 How to Create a Global JDBC Data Source

To run an ADF application that uses JPA entities you first need to create a global JDBC data source. You can do this in by running the GlassFish admin console in the Eclipse browser.

Once you have created the JDBC connection pool, you can define a JDBC resource that uses your new connection pool.

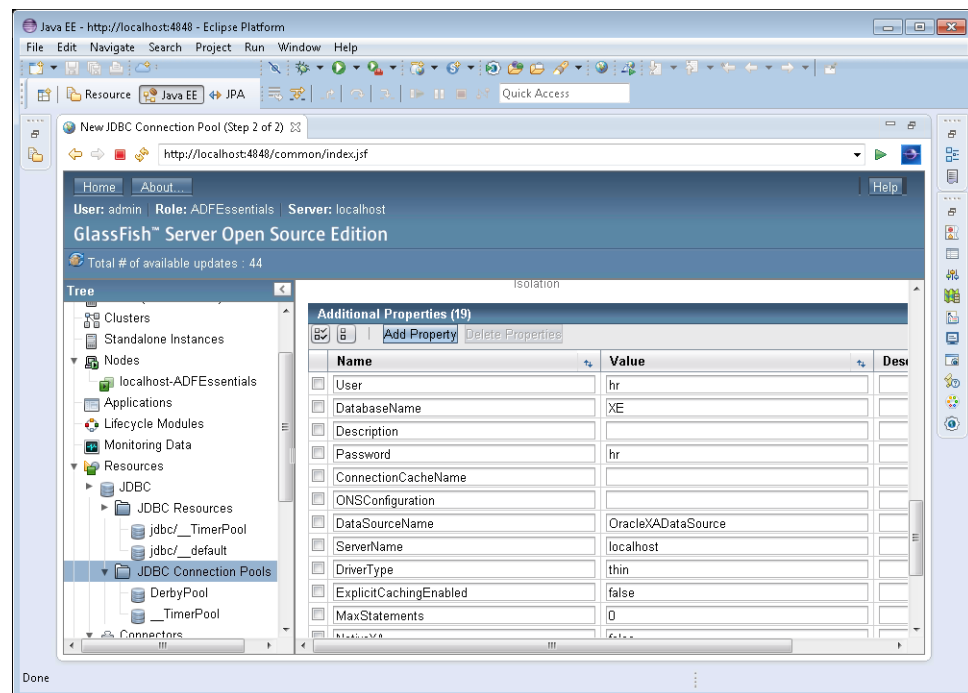
To create a Global JDBC Data Source

1. In OEPE from the Server view select GlassFish and right-click and choose **GlassFish > View > Admin Console**.

The GlassFish admin server opens in the Eclipse browser.

2. Expand the Resources node and choose **JDBC > JDBC Connection Pools**. Click **New** to open the new JDBC Connection Pool page.
3. Name your connection, choose **Resource Type** of `javax.sql.XADataSource`, and choose the **Database Driver Vendor** `Oracle`. Make sure that **Introspect** is unselected (introspection not enabled). Click **Next**.
4. In New JDBC Connection Pool (step 2 of 2), scroll to Additional Properties and add your database information, as shown in [Figure 2-26](#).

Figure 2-26 *GlassFish Server Console - Additional Properties*



For example, the values for Oracle XE running on the local machine are:

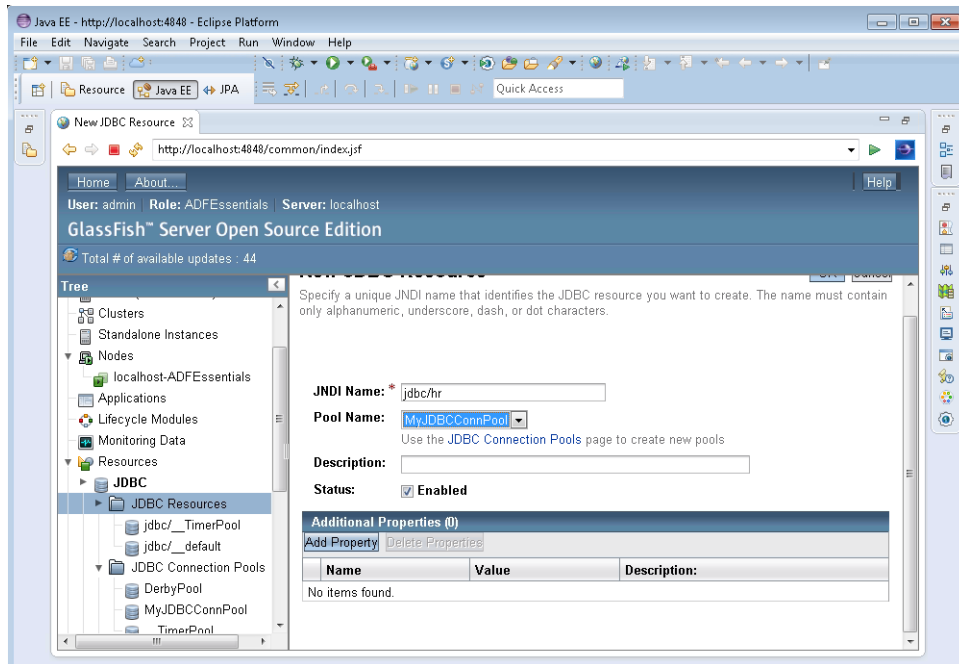
- user = hr
- Password = hr
- databaseName = XE
- ServerName = localhost
- DriverType = thin

- PortNumber =1521
5. Click **Finish**. Check your connection by selecting the new JDBC connection pool you have just created in the Tree, and clicking **Ping** on the General tab. If your ping is successful, you next need to define a JDBC resource that uses your new JDBC connection pool.

To define a JDBC resource that uses your new connection pool

1. In the admin console, go to **Resources > JDBC > JDBC Resources>**. Click **New** to display the New JDBC Resource page, as show in [Figure 2–27](#).

Figure 2–27 GlassFish Server Console - New JDBC Resource



2. Enter the JNDI name, for example jdbc/hr.
3. Select your new connection pool.
4. Click **OK**.

2.10.8 Known Problems and Solutions

The following are issues that you might encounter with your GlassFish installation and runtime, and some solutions to address these issues.

1. **ADF DVT components are not rendered.**

If your ADF DVT components such as Graph are not rendered at runtime, the issue is that the application is missing the `dvt-as.jar` and `dvt-shared-js.jar`. Do the following steps to correct this issue:

1. Download the latest distribution of ADF Essentials and extract the JARs as detailed in [Section 2.10.1, "How to Download ADF Essentials"](#).
2. Update the libraries in the Workspace User Library, as detailed in [Section 2.10.5, "How to Register the ADF Essentials Client WAR Library in Your Workspace"](#).

3. Remove the deployed application from the server and re-deploy the application.
2. **Content on a page failing to load.**

If you get the error message "The content of this page failed to load as expected because data transmission was interrupted. Please try again or contact your system admin", follow the steps above for [ADF DVT components are not rendered](#).

3. **Running the application gives the HTTP Status 500 -ADF_Faces-30200 error:**

If you get the error message "HTTP Status 500 - ADF_FACES-30200:For more information, see the server's error log for an entry beginning with: The UIViewRoot is null. Fatal exception during PhaseId: RESTORE_VIEW 1."

Do the following steps:

1. Un-deploy the application.
 2. Clean the server instance. With the server stopped, right-click and choose Clean.
 3. Re-deploy the application.
 4. **Unable to deploy an application that includes JPA Entities.**
- If you have trouble deploying your application that includes JPA entities, do the following steps:
1. Define a JDBC DataSource in GlassFish, as described in [Section 2.10.7, "How to Create a Global JDBC Data Source"](#).
 2. Add the data source you just defined to your JPA `persistence.xml`.

2.11 Appendix A Oracle ADF XML Files

For more information about configuring GlassFish Server for Oracle ADF Essentials, see "Configuring GlassFish Server" in *Administering Oracle ADF Applications*.

Metadata files in an Oracle ADF application are structured XML files used by the application to:

- Specify the parameters, methods, and return values available to your application's Oracle ADF data control usages.
- Define configuration information about the UI components in JSF and ADF Faces.
- Define application configuration information for the Java EE application server.

The ADF metadata files created by OEPE are listed below.

2.11.1 Oracle ADF Data Binding Files

OEPE creates the following ADF data binding files:

- `adfm.xml`: This file lists the `DataBindings.cpx` file that is available in the current project. For more information about `adfm.xml`, see [Section A.4, "adfm.xml" in *Developing Fusion Web Applications with Oracle Application Development Framework*](#).
- `DataBindings.cpx`: This file contains the page map, page definitions references, and data control references. The file is created the first time you create a data binding for a UI component. The `DataBindings.cpx` file defines the Oracle ADF binding context for the entire application. The binding context provides access to

the bindings and data controls across the entire application. See Section A.7, "DataBindings.cpx" in *Developing Fusion Web Applications with Oracle Application Development Framework*.

- `pagenamePageDef.xml`: This is the page definition XML file. It associates web page UI components with data, or data controls. OEPE creates this file each time you design a new web page. These XML files contain the metadata used to create the bindings that populate the data in the web page's UI components. For every web page that refers to an ADF binding, there must be a corresponding page definition file with binding definitions. See Section A.8, "pagenamePageDef.xml" in *Developing Fusion Web Applications with Oracle Application Development Framework*.

2.11.2 Web Configuration Files

OEPE creates the following Oracle ADF web configuration files:

- `web.xml`: Part of the application's configuration is determined by the contents of its Java EE application deployment descriptor, `web.xml`. The `web.xml` file defines everything about your application that a server needs to know. The file plays a role in configuring the Oracle ADF data binding by setting up the `ADFBindingFilter`. Additional runtime settings include servlet runtime and initialization parameters, custom tag library location, and security settings.

An ADF Faces application typically uses its own set of configuration files in addition to `web.xml`. For more information, see the "Configuration in `trinidad-config.xml`" section in *Developing Web User Interfaces with Oracle ADF Faces*.

- `adfc-config.xml`: The configuration file for an ADF unbounded task flow. The configuration file contains metadata about the activities and control flows contained in the unbounded task flow. The default name for this file is `adfc-config.xml`, but an end-user can change the name.
- `task-flow-definition.xml`: The configuration file for an ADF bounded task flow. The configuration file contains metadata about the activities and control flows contained in the bounded task flow. The default name for this file can be `task-flow-defintion.xml` or whatever an end user specifies when creating the task flow. The same application can contain multiple task flow definition files.

For more information on Oracle ADF configuration files, see the Oracle ADF XML Files appendix in *Developing Fusion Web Applications with Oracle Application Development Framework*.

Oracle MAF Tools Support

Oracle Enterprise Pack for Eclipse (OEPE) provides a set of plug-ins for the Eclipse IDE designed to create, configure, and run Oracle Mobile Application Framework (MAF) applications.

This document contains the following section about using Mobile Application Framework with OEPE:

- [Developing with Oracle MAF](#)

3.1 Developing with Oracle MAF

Oracle Enterprise Pack for Eclipse (OEPE) supports Oracle Mobile Application Framework (MAF), a solution that enables you to create mobile applications that run natively on both iOS and Android phones and tablets.

For more information, see *Developing Mobile Applications with Oracle Mobile Application Framework (OEPE Edition)*.

Oracle WebLogic Server Support

Oracle WebLogic Server Tools is a set of plugins for the Eclipse IDE designed to help develop, deploy, and debug applications for Oracle WebLogic Server.

This document contains the following sections:

- [Feature Overview](#)
- [WebLogic Shared Libraries](#)
- [Support for WebLogic SCA](#)
- [Support for WebLogic Scripting Tool \(WLST\)](#)
- [Editing Deployment Descriptors](#)
- [Using Deployment Plan Editor](#)

4.1 Feature Overview

The information in this section describes how to use WebLogic Server with the Eclipse IDE.

The following versions of Oracle WebLogic Server are supported:

- 12c Release 1 (12.1.3)
- 12c Release 1 (12.1.2)
- 12c Release 1 (12.1.1)
- 11g Release 1 (10.3.6)
- 11g Release 1 (10.3.5)
- 11g Release 1 (10.3.4)
- 11g Release 1 (10.3.3)
- 11g Release 1 (10.3.2)
- 11g Release 1 (10.3.1)
- 10g Release 3 (10.3.0)
- 10.0
- 9.2

4.2 WebLogic Shared Libraries

A WebLogic Shared Library is an Enterprise Application Archive, a stand-alone EJB, a Web Application module, or a JAR file that is registered with Oracle WebLogic Server as a shared library. The library resources can be shared between multiple applications, alleviating the need to have duplicate copies of the resources in each application. For an overview of WebLogic Shared Libraries, see "Deploying Shared Java EE Libraries and Dependent Applications" in *Deploying Applications to Oracle WebLogic Server*.

WebLogic shared libraries provide an easy way to share one or more types of Java EE modules among multiple applications.

A shared library can be one of the following:

- Stand-alone EJB module
- Stand-alone Web application module
- Multiple EJB modules packaged in an EAR file
- Multiple Web application modules packaged in a WAR file
- Single JAR file that is registered with the application container

After the library has been registered, you can deploy multiple applications that reference the library. Each referencing application can use the library as if it were packaged as part of the referencing application itself. The shared library classes are added to the classpath of the referencing application, and the referencing application's deployment descriptors are merged (in memory) with those of the library.

A registry of shared libraries can be viewed and edited using the workspace Preference page at **Window > Preferences > WebLogic > Shared Libraries**. At run time, the registry is described in the domain's `config.xml` file.

These are terms used when working with WebLogic Shared Libraries:

Library reference: WebLogic Shared Libraries are referenced indirectly by specifying the name of the library, the specification version (optional), the implementation version (optional) and whether a newer version of the library should be used if present. The library references are used on the project classpath and in the deployment descriptors (`weblogic-application.xml` and `weblogic.xml`). Library references are resolved against the libraries registry when necessary.

Libraries registry (or registry): The list of known WebLogic Shared Libraries which is used to resolve a library reference.

4.2.1 Common Operations

This section describes the following common operations associated with WebLogic shared libraries.

4.2.1.1 Adding a New Library to the Registry

You can add new libraries to the registry.

To add a new library to the registry:

1. From the top-level menu, select **Window > Preferences**.
2. Find the WebLogic node in the tree on the left-hand-side of the dialog.
3. Select **WebLogic > Shared Libraries** from the list of preferences on the left.
4. Click **Add**.

5. Click **Browse** to select the library location.
6. Find the library archive where it is located on the disk, and select the library.
7. Verify the information presented in the Attributes section of the Add WebLogic Shared Library dialog. You can modify the library name, the specification version, and implementation version by using the Attributes table as long as they are not specified in the library's `manifest.mf` file.
8. Click **OK**.

4.2.1.2 Adding a Library Reference to the Project Classpath

You can add a library reference to the project classpath.

To add a library reference to the project classpath:

1. Right-click on the project in the Project Explorer view and select **Properties** from the drop-down menu. This will open the Properties dialog.
2. In the Properties dialog, select the **Java Build Path** from the list of properties.
3. In the Java Build Path part of the dialog, select the Libraries tab.
4. Click **Add Library**.
5. Select **WebLogic Shared Library**, and then click **Next**.
6. Click **Browse**, and choose the library that you want to reference.
7. Modify the reference information, if necessary.
8. Click **Finish**.

4.2.1.3 Modifying a Library Reference on the Project Classpath

You can modify library references.

To modify a library reference on the project classpath:

1. Select the project in the Project Explorer view.
2. Choose **Project > Project Properties**.
3. Locate the Java Build Path node in the tree on the left-hand-side of the dialog and select it.
4. Open the Libraries tab.
5. Find the entry in the list of libraries called **Shared Library *library-name*** and select it.
6. Click **Edit**.
7. Review the library reference. If any changes are necessary, make the changes and click **Finish**.

4.2.1.4 Removing a Library Reference from the Project Classpath

You can remove library references.

To remove a library reference from the project classpath:

1. On the Properties dialog, select the **Java Build Path** from the list of properties.
2. On the Java Build Path part of the dialog, select the Libraries tab.
3. Find the entry in the list of libraries called **Shared Library *library-name***, and then select it.

4. Click **Remove**.

4.2.2 Validation Problems

This section lists error and warning messages that may appear in the Problems view. Resolutions are provided for each error/warning message.

4.2.2.1 Validation Errors

- Unable to resolve library reference on project classpath
- Unable to resolve library reference in the deployment descriptor
- Library on classpath but not in weblogic-application.xml
- Library on classpath but not in weblogic.xml
- Web library referenced by a non-Web project
- Non-Web library referenced in weblogic.xml
- Project must be part of an EAR to use this library
- Project must be part of a web application to use this library

Unable to resolve the reference to "<library-name>" library on project classpath.

Problem: The library reference to a WebLogic Shared Library on the project classpath cannot be resolved using the workspace libraries registry. The project cannot be build.

Resolution 1: Add the appropriate library to the libraries registry. For instructions on how to do this, see [Adding a new library to the registry](#).

Resolution 2: Modify the library reference to match what is present in the registry. For instructions on how to do this, see [Modifying a library reference on the project classpath](#).

Resolution 3: Remove the library from the project classpath. For instructions on how to do this, see [Removing a library reference from the project classpath](#).

Unable to resolve the reference to "<library-name>" library in the deployment descriptor.

Problem: The library reference to a WebLogic Java EE Library in the deployment descriptor (the `weblogic.xml` or `weblogic-application.xml` file depending on project type) cannot be resolved using the workspace libraries registry. The project may not run once deployed.

Resolution: Add the appropriate library to the libraries registry. For instructions on how to do this, see [Section 4.2.1.1, "Adding a New Library to the Registry"](#).

"<library-name>" library is on the classpath of this project, but is not in the weblogic-application.xml file of the EAR project "<EAR-project-name>"

Problem: The project contains a reference to a WebLogic Shared Library, but the `weblogic-application.xml` file of the EAR project that this project belongs to does not reference the library. This could lead to problems at run time since Oracle WebLogic Server will not know how to make this library available for use by this module.

Resolution: Remove the library from the project classpath. For instructions on how to do this, see [Section 4.2.1.4, "Removing a Library Reference from the Project Classpath"](#).

"<library-name>" library is on the classpath of this project, but is not in the weblogic.xml file.

Problem: The project contains a reference to a WebLogic Shared Library, but the weblogic.xml file does not reference this library. This could lead to problems at run time since Oracle WebLogic Server will not know how to make this library available for use by this module.

Resolution: Remove the library from the project classpath. For instructions on how to do this, see [Section 4.2.1.4, "Removing a Library Reference from the Project Classpath."](#)

"<library-name>" library is a web library and cannot be referenced by this project.

Problem: Oracle WebLogic Server only allows classes in a WAR-type WebLogic Shared Library to be visible to Web modules. While an EAR can reference such a library in the weblogic-application.xml, none of the classes in the referenced library will be visible to the classes in the EAR.

Resolution: Remove the library from the project classpath. For instructions on how to do this, see [Section 4.2.1.4, "Removing a Library Reference from the Project Classpath."](#)

"<library-name>" library reference is not allowed in the weblogic.xml file. Only WAR libraries are allowed.

Problem: Oracle WebLogic Server only supports WAR libraries to be referenced from the weblogic.xml deployment descriptor. A non-WAR library has been referenced in the descriptor.

Resolution: Remove the library from the weblogic.xml deployment descriptor.

This project must be part of an EAR in order to use "<library-name>" library.

Problem: The library referenced on the classpath of this project can only be accessed at run time if this project is part of an EAR and the EAR's weblogic-application.xml deployment descriptor references this library.

Resolution 1: Remove the library from the project classpath. For instructions on how to do this, see [Section 4.2.1.4, "Removing a Library Reference from the Project Classpath."](#)

Resolution 2: Create a new EAR project (or use an existing one). In the EAR project's Properties (right-click the EAR and select **Properties**) select the page labeled **Java EE Module Dependencies**. Select the project from the list, and then click **Finish** to associate the project with the EAR project.

This project must be part of a web application in order to use "<library-name>" library.

Problem: The library referenced on the classpath of this project can only be accessed at run time if this project is part of a Web application, and the Web application's weblogic.xml deployment descriptor references this library.

Resolution 1: Remove the library from the project classpath. For instructions on how to do this, see [Section 4.2.1.4, "Removing a Library Reference from the Project Classpath."](#)

Resolution 2: Create a new Dynamic Web project (or use an existing one). In the Web project's Properties (right-click the Web project, and select Properties) select the page labeled **Java EE Module Dependencies**. Select the **Web Libraries** tab. Select the project from the list, and then click **Finish** to associate the project with the Web project.

4.2.2.2 Validation Warnings

This section lists warning messages that may appear in the Problems view. Resolutions are provided for each warning message.

- Classpath reference differs from weblogic-application.xml
- Classpath reference differs from weblogic.xml

The reference to the "<library-name>" library on this project's classpath differs from the reference to this library in the weblogic-application.xml file of the EAR project "<EAR-project-name>".

Problem: The library reference on the module project's classpath differs from the library reference in the weblogic-application.xml file of the EAR project that this module project belongs to. This could lead to problems at run time, because the project could be compiled against a different version of the library than what will be used at run time.

Resolution: Modify the library reference on the project classpath so it matches the library reference in the EAR project's weblogic-application.xml file. For instructions on how to do this, see [Section 4.2.1.3, "Modifying a Library Reference on the Project Classpath."](#)

The reference to the "<library-name>" library on this project's classpath differs from the reference to this library in the weblogic.xml file.

Problem: The library reference on this project's classpath differs from the library reference in the weblogic.xml file. This could lead to problems at run time, because the project could be compiled against a different version of the library than what will be used at run time.

Resolution: Modify the library reference on the project classpath so it matches the library reference in the weblogic.xml file. For instructions on how to do this, see [Section 4.2.1.3, "Modifying a Library Reference on the Project Classpath."](#)

4.3 Support for WebLogic SCA

OEPE provides support for WebLogic SCA container. You use it to populate the Spring context file and bundle as part of any regular Java EE deployment bundles, such as EAR or WAR.

Note: Prior to using OEPE support for WebLogic SCA in your Eclipse project, you need to configure Oracle WebLogic Server 11g Release 1 (10.3.2) or later at localhost.

4.3.1 Configuring Projects to Use WebLogic SCA

You enable your project for WebLogic SCA by adding a facet provided by OEPE.

To configure a project:

1. Create a dynamic Web project by right-clicking the Project Explorer and choosing **New > Dynamic Web Project**. This opens the New Dynamic Web Project dialog:
 - Specify the name for your project.
 - Set the dynamic Web module version.
 - Set the target runtime to Oracle WebLogic Server 11gR1 (10.3.2) or later.

- Select **Add project to an EAR** and provide a new EAR name.
 - Click **Modify** on the Configuration field to open the Project Facets page of the Properties dialog:
 - Ensure that **WebLogic Web App Extensions** facet is selected.
 - Select **Spring** facet with version 2.0 or later.
 - Select **WebLogic SCA** facet, and then click **OK** to close the Project Facets dialog.
2. Click **Next** on the New Dynamic Web Project dialog.
 3. You may accept or modify default settings on the next New Dynamic Web Project > Web Module screen, and then click Next. This opens the New Dynamic Web Project > Spring screen.
 4. If you do not see Spring Framework 2.5.6 listed, click **Download Library**. On the Download Library dialog, select **Spring Framework 2.5.6** library provided by Oracle, and then click **Next**. Accept the terms of the Apache License, and then click **Finish**.

Note: If your machine is located inside of a network which requires a proxy to access outside resource such as the Internet, the download may fail due to the fact that Eclipse IDE includes a Web browser to let you access the Internet from within the IDE. In this case, reconfigure your Eclipse IDE proxy settings using Window > **Preferences** > **General** > **Network Connections**, and try again.

5. Upon the completion of the Spring library download, make sure the Spring Framework 2.5.6 library is selected on the **New Dynamic Web Project > Spring** dialog. You may choose to select **Create a Spring bean definitions file** to generated the `applicationContext.xml` file and trigger the update of `web.xml` file to load the Spring bean configuration file, and then click **Next**.
6. Click **Finish** on the **New Dynamic Web Project > WebLogic SCA** screen that to complete your configuration.

Upon the completion of the configuration, OEPE adds WebLogic SCA shared library and Spring library to your project, as well as creates the `spring-context.xml` file in your project's `META-INF/jsca` directory.

You can open the `spring-context.xml` file either in source view by double-clicking the file name in the Project Explorer, or in graphical view by right-clicking the file name and selecting **Open Graph** from the drop-down menu.

You use the `spring-context.xml` file to define new beans, specify services, bindings, and so on. OEPE provides templates to facilitate these definitions. For more information, see [Section 4.3.3, "Creating Complex Properties Using XML Template."](#)

4.3.2 Using Context Help for WebLogic SCA XML Attributes

OEPE provides context help for each SCA XML element and attribute. To access help topics, hover the mouse over elements in the XML editor.

4.3.3 Creating Complex Properties Using XML Template

OEPE provides a set of XML templates that you can use to populate a predefined group of XML elements. This facilitates creating of complex properties in binding specification, such as, for example, the `PolicyReference`.

To invoke an XML template, you type the name of the template (for example, "SCA"), and then press **CTRL+Space**.

You can use the following XML templates for creating WebLogic SCA definitions:

- SCA service with Web services binding
- SCA service with EJB binding
- SCA reference with Web services binding
- SCA reference with EJB binding

Inside of WebLogic SCA Service with Web services binding, you can invoke the `PolicyReference` template to further customize the binding security setting, as follows:

- `PolicyReference`: Username token with message protection: WSS 1.0 X509 with asymmetric binding.
- `PolicyReference`: Username token with message protection: WSS 1.1 symmetric binding and authentication with plain-text Username Token which is encrypted and signed using the Symmetric key.
- `PolicyReference`: X509 certificate authentication with message protection (WSS 11).
- `PolicyReference`: Anonymous with message protection (WSS 11).
- `PolicyReference`: ID Propagation using SAML token [sender-vouches] with message protection (WSS 11).
- `PolicyReference`: Username token over SSL.
- `PolicyReference`: SAML token (Sender Vouches) over SSL.

Applicable to both WebLogic SCA Service and SCA Reference, the Property template is also available to further customize the binding security setting, as follows:

- Property: SDO schema file location.
- Property: External customization file.
- Property: WSDL cache timeout. Note that this Property (`weblogic.sca.binding.ws.referenceWsdLCacheTimeoutMins`) applies to references only.
- Property: Enable using OwsM policies.

4.3.4 Creating WebLogic SCA Data-Binding Customization Descriptor

With OEPE, you can create WebLogic SCA data-binding customization descriptor for which the schema file is bundled. This descriptor defines the external mapping metadata for the data-binding framework. The data is used to define the attributes of a particular Java Web service endpoint interface. Each change that you make to the XML will be validated against the schema.

To create a WebLogic SCA data-binding customization descriptor:

1. In the Project Explorer, right-click a Java directory in within your Web project and select **New > Other** from the drop-down menu to open the Select a wizard dialog.

2. From the list of available wizards, select **WebLogic Configuration Files > WebLogic SCA Databinding Customization Descriptor**, and then click **Next**.
3. In the New WebLogic SCA Databinding Customization Descriptor dialog, provide the name for the descriptor, specify the Java type by completing the **Select Java Type**, then click **Finish**.

The `databinding_cust.xml` file is added to your project.

4.3.5 Deploying a WebLogic SCA Application

You use Oracle WebLogic Server 11g Release 1 (10.3.2) or later to deploy your dynamic Web project configured for WebLogic SCA.

To deploy the dynamic Web project:

1. Start an instance of Oracle WebLogic Server 11g Release 1 (10.3.2) or later at `localhost`.
2. Open the server configuration (**Overview**) page by double-clicking on the server name in the Servers view. Ensure that the information is correct. Notice that the domain area does not contain your project's name.
3. Add your configured project's EAR to the server. To do so, right-click Oracle WebLogic Server 12c (12.1.1) at `localhost` in Servers view, select **Add and Remove Projects** from the drop-down menu, find your the EAR in the list of projects, and then click **Add** followed by **Finish** on the Add and Remove Projects dialog.

Now the server configuration page displays the project and its EAR under the `base_` domain.

The server console displays the deployment logging information.

4.3.6 Running a WebLogic SCA Application

After deploying the application, you can run it.

To run the application:

- In the Project Explorer, right-click a `sca:service` node listed under beans in the `spring-context.xml` file and choose **Run As > Run On Server**.

Upon completion of the Run On Server dialogs, a WSDL file, which you can use with other OEPE features, is created by the WebLogic SCA deployment process and loaded in a browser.

4.4 Support for WebLogic Scripting Tool (WLST)

WLST is a scripting tool for monitoring, managing, and configuring Oracle WebLogic Server from the command line. WLST is based on Jython programming language with WebLogic WLST libraries. The execution can happen in the following three modes: scripting, interactive, and embedded. WLST can be enabled for online and offline connection modes and can act as a JMX client.

OEPE provides tooling for WLST that enable editing, executing, debugging, WebLogic MBean access and navigation, as well as a built-in help for WLST commands.

4.4.1 Configuring Projects for WLST


Using OEPE, you can add WLST facet to your Java projects that run on Oracle WebLogic Server. Note that Utility projects are most suitable for this functionality.

To configure your project for WLST:

1. Either add the WLST facet when you create a faceted project, or add the facet to an existing project by right-clicking your project in the Project Explorer and selecting **Properties** from the drop-down menu. This opens the Properties dialog.
2. In the Properties dialog, select **Project Facets** on the left panel, and then select **WLST** from the Project Facet list.
3. Optionally, click **Further configuration available** to open the Configure WebLogic Scripting Tools dialog. This dialog allows you to configure WLST script source path and targeted runtime. When you have finished, click **OK**.
4. To complete adding WLST support to your project click **Apply > OK** on the Properties dialog.

4.4.2 Creating New WLST Files

You can create a new WLST script as follows:

1. In the Project Explorer, right-click your WLST-enabled project and select **New > Other** from the drop-down menu.
2. On the New dialog, select **WebLogic > WLST Script** and then click **Next** to open the WLST Script dialog.
3. Specify the location for the new WLST file.
4. To specify the template to use, click **Browse**  to open the TemplateName dialog, where you can choose one of the standard templates such as a default one, or specify a custom template of a module scope that you defined yourself. For more information, see [Section 4.4.4, "Adding WLST Templates."](#) Click **OK**.
5. In the WLST dialog click **Finish**.

A new WLST script file is created in the specified location. You can execute this file, as well as edit it using either the source editor, or a WLST file editor provided by OEPE. For more information, see [Section 4.4.7, "Executing WLST"](#) and [Section 4.4.3, "Editing WLST Script."](#)

4.4.3 Editing WLST Script

OEPE provides an editor for WLST files. The editor features the following:

- Syntax highlighting.
- Code completion for Jython and WLST built-in functions.
- WLST Help view that includes detailed WLST command reference materials.

To edit a WLST script:

- In the Project Explorer, double-click the WLST file. It opens in the editor.

4.4.4 Adding WLST Templates

OEPE provides templates that you can use to create WLST files.

To add a new WLST template:

1. Select **Window > Preferences** from the top-level menu to open the Preferences dialog.

2. On the Preferences dialog expand the **Pydev > Editor** node, and then select **Templates** to open the list of available templates.
3. Click **New** on the Preferences > Templates screen to open the New Template dialog.
4. Enter the following information about your new template:
 - The template's name.
 - The template's scope: editor or module. If you select the module scope, your new template will appear in the selection list of templates when you create a new WLST script.
 - A brief description.
 - A pattern on which to base the template. You can define a custom pattern, or select one from the list of predefined patterns by clicking **Insert Variable**.
5. Click **OK** to close the completed New Template dialog.
Notice that your new template is now listed in the Preferences > Templates dialog.
6. Click **Apply > OK** on the Preferences > Templates dialog.

4.4.5 Navigating MBean Structures

When writing WLST script, you often need to navigate WebLogic runtime MBean structures. OEPE enables you to view the MBean hierarchy in the Servers pane.

To add or generate WLST code from MBean Hierarchy:

1. Open your WSLT file.
2. Click the Servers tab to open the Servers pane.
3. Start WLS by choosing **Start** from the context menu.
4. In the Servers view, expand the MBean Hierarchy node.
5. Right-click any child node and select **Generate WSLT Code** from the context menu.

Note: This menu item is enabled only when the WSLT file is in focus.

4.4.6 Using WLST Console

You can launch WLST in interactive mode in the Console view. This allows you to manage the Oracle WebLogic Server life cycle, monitor the server status, and prototype WLST script using WLST commands in the command line.

You use WLST console, activate the Console tab, and then select **WLST > Oracle WebLogic Server Version** from the tab menu.

To interact with the server, type WLST commands, for example `help()`.

To stop WLST, click **Terminate** on the Console menu.

4.4.7 Executing WLST

To execute the WLST script, right-click the file in the Project Explorer, and select **Run As > WLST Run**.

WLST will be launched and the output displayed on the Console view.

4.4.8 Debugging WLST Script

To debug the WLST script, right-click the file in the Project Explorer, and then select **Debug As > WLST Run**.

WLST debugger will start and the output will be displayed on the Console view.

The Pydev debugger allows you to do the following:

- Set break point in the WLST source file.
- Step over.
- Examine variables using the Variables view.

Note: The WLST script is executed with Jython interpreter in debug mode, as opposed to the `weblogic.WLST` interpreter.

4.4.9 Importing Existing WLST Script into OEPE

If you import `wlstModule` into default name space using the `from wlstModule import *` statement, the global object `cmo` will not be available in this mode. You can work around this condition by either importing `wlstModule` with its own namespace:

```
import wlstModule as wl
```

or by creating a local `cmo` object from the return of `cd()` statement:

```
cmo=cd( MBEAN_PATH )
```

When importing existing WLST scripts into OEPE, be aware of the fact that Eclipse may flag WLST commands with "undefined variable" error.

To work around this problem, add following conditional import statement in the beginning of the file:

```
if __name__ == '__main__':  
    from wlstModule import *
```

This statement is intended to help the builder to locate the WLST command. The `wlstModule` will be imported only when the script is executed with Jython interpreter.

4.4.10 Known Issues and Limitations

If you use WebLogic Server 9.2 on Windows Vista or Windows 7 operating system, you should avoid using Jython variable `os.environ` in your WLST script. Instead, you have to manually change the value of `BEA_HOME` and `WL_HOME` variables in the generated WLST script.

4.5 Editing Deployment Descriptors

OEPE provides a graphical design view that you can use to edit Oracle WebLogic Server-specific deployment descriptors in the form of `weblogic.xml`, `weblogic-application.xml`, `weblogic-ejb-jar.xml`, and `WV-jms.xml` files.

4.5.1 Using Deployment Descriptor Editors

To open your deployment descriptor files in the editor, either double-click on the file name, or right-click the file and select one of the following from the drop-down menu:

- **Open With > WebLogic Web Module Deployment Descriptor Editor** to open `weblogic.xml`.
- **Open With > WebLogic Application Deployment Descriptor Editor** to open `weblogic-application.xml`.
- **Open With > WebLogic Application Client Deployment Descriptor Editor** to open `weblogic-application-client.xml`.
- **Open With > WebLogic EJB Jar Editor** to open `weblogic-ejb-jar.xml`.
- **Open With > WebLogic JDBC Configuration Editor** to open `weblogic-jdbc.xml`.
- **Open With > JMS Descriptor Editor** to open `WV-jms.xml`.

You can specify values, as well as enable or disable various descriptor elements by making selections on the **Outline** pane, and then modifying values on the right side of the editor. To obtain information about fields and possible settings for each of them, consult the online help by moving the focus to the subject field, and then pressing F1.

Note: The setting in the Server version field in the General section of the editor is particularly important to the rest of the editor: some functionality in the editor may be enabled or disabled depending on which Oracle WebLogic Server version you specify, so you must be accurate in setting your target environment.

The descriptor editor has the following features:

- Every field is enabled for online help, which you can access by moving the focus to the subject field and pressing F1.
- There is a field-level validation on every field.
- Clicking on a linked field name takes you to the element or object (such as a class, for example) that is entered in this field.
- You can restore default settings by clicking the **Restore Defaults** button located at the top right corner of the editor.
- Table cells are equipped with a cell editor: when working with tables in the editor, double-clicking on a cell activates a cell editor. Some cells have a browse button when the cell is in the edit mode.
- A limited keyboard navigation is enabled on the editor.

4.5.1.1 Editor Keyboard Navigation

The deployment descriptor editors allow you to use the following keys or key combinations to perform some of the operations that can also be done through mouse actions.

Tab Navigates through editor fields.

Arrow Keys Navigate through the content outline and tables.

ESC Closes many of the dialogs opened by the editor, including the property editor assistance popup.

Enter Selects items in combo boxes and deactivates cell editors in tables.

Ctrl+L Opens the Browse dialog when focus is on a browsable field.

Ctrl+I Opens the property editor assistance popup.

Ctrl+J Jumps to the entity referenced by the value contained in the current field. The value can be a class name, a file path, etc.

Ctrl+Up Moves the selected entity towards the head of the list by one position.

Ctrl+Down Moves the selected entity towards the tail of the list by one position.

4.5.2 Creating JMS Descriptors

You can create a JMS descriptor for Oracle WebLogic Server by following this procedure:

1. Ensure that the EAR project to which you are planning to add a JMS descriptor has a WebLogic EAR Extensions facet enabled. If the facet is disabled, enable it as follows:
 - a. Right-click the project in the Project Explorer.
 - b. Select **Properties** from the drop-down menu.
 - c. Select **Project Facets** from the list of properties.
 - d. Select **WebLogic EAR Extensions**.
 - e. Click **Apply** and **OK** to finalize your configuration.
2. In the Project Explorer, right-click your EAR project and select **New > Other** from the drop-down menu. On the New dialog, select **WebLogic Descriptors > WebLogic JMS Descriptor**, and then click **Next**. This will open the New WebLogic JMS Descriptor > File Name and Location dialog.
3. On the New WebLogic JMS Descriptor > File Name and Location dialog, set the following:
 - select the parent folder for your descriptor;
 - make an appropriate selection in the Register a corresponding JMS module in weblogic-application.xml field: if you enable this option, the following code will be added to your project's weblogic-application.xml source file:

```
<wls:module>
  <wls:name>module-jms</wls:name>
  <wls:type>JMS</wls:type>
  <wls:path>../module-jms.xml</wls:path>
</wls:module>
```

4. Click **Finish**.

Upon completion, your new JMS descriptor opens in the appropriate editor view.

4.6 Using Deployment Plan Editor

Using a deployment plan, you can optionally define or override Oracle WebLogic Server tuning parameters to optimize the use of resources in the target environment.

4.6.1 Creating a New Deployment Plan

Using OEPE, you can create a new deployment plan by following this procedure:

1. Open the Java EE perspective by selecting **Window > Open Perspective > Other > Java EE** from the Eclipse IDE main menu.

2. Right-click the Project Explorer and select **New > Other** from the drop-down menu. On the New dialog, select **WebLogic Descriptors > WebLogic Deployment Plan**, and then click Next. This opens the File Name and Location dialog.
3. On the File Name and Location dialog, select the parent folder and provide the name for your deployment plan, and then click **Next**.
4. On the next Target Application and Options screen, select the target application, target Oracle WebLogic Server runtime, optionally specify whether or not to invoke `weblogic.PlanGenerator` to generate default variable definitions, and then click **Finish**.

For more information about `weblogic.PlanGenerator`, see "weblogic.PlanGenerator Command Line Reference" in *Deploying Applications to Oracle WebLogic Server*.

Upon the completion, the New WebLogic Deployment Plan wizard opens the deployment plan in the deployment plan editor for further modifications.

4.6.2 Editing a Deployment Plan

OEPE provides a graphical design view that you can use to edit your WebLogic deployment plan.

The editor consists of the following parts:

- A tree control on the left.
- An edit page on the right - when you select a node in the tree, the editor will display an appropriate edit page.

The tree control is represented by the following main nodes:

- **General** - lets you modify the application name, server and deployment plan versions, the external configuration root, and provide the deployment plan description.
- **Variable Definitions** - lets you define new variables. To create a new variable, select the Variable Definitions node, and then click + (Add) on the edit page to expand the area
- **Modules** - lets you specify existing or add new modules by clicking Add a module override.

Click **Browse** next to the Module Name field to open the Module Name dialog. Select an existing module or add a module override to the deployment plan.

To add a deployment descriptor, click **Add a descriptor** on the Modules editor page.

Specify the descriptor URI by clicking **Browse** next to the URI field on the Descriptor editor page. This will display the Select the URI of the descriptor dialog,.

Using the Select the URI of the descriptor dialog, select the `weblogic.xml` file, and then click **OK**.

To add a new variable assignment, click + (Add) on the **Variable Assignments > Specify the XPath** area.

Use the XPath Expression Builder dialog to navigate the element that you want to override in the descriptor, and then click **OK** to select the XPath.

Click **Browse** next to the **Variable name** field to display the Variable Name dialog and select the variable name to assign.

Select the **Replace** operation on the Variable Assignments edit page.

Use the **Clean Up Deployment Plan** utility to remove unassigned variables from your plan's XML file. To do so, click the **Clean Up** button represented by a green check mark icon located in the top right corner of the editor. Make your selection from the list of the potential clean up candidates, and then click **Finish** to remove these elements from your deployment plan.

You can also switch to the source view of your WebLogic deployment plan to edit it manually.

4.6.3 Using an Existing Deployment Plan to Configure an Application

You can use an existing deployment plan from the command line, or from the IDE.

To use a generated deployment plan from the command line use the following syntax in a prompt:

```
java weblogic.Deployer -adminurl t3://localhost:7001 -username weblogic -password weblogic7001 -plan plan.xml -deploy MyNewWSProjectEAR.ear
```

To deploy your application from the IDE to WebLogic either locally or remotely, go to **Server Properties > WLS**, expand **WebLogic Publishing** and choose **Advanced**.

Integrating Oracle Cloud Services

Oracle Cloud Service enables you to use the cloud to develop, collaborate, and deploy your OEPE applications from one central source.

Currently OEPE supports only the Java Cloud Service, which includes the WebLogic server, and optional Oracle Cloud database services.

For more information about Oracle Cloud, or to open an Oracle Cloud account, visit <http://cloud.oracle.com>.

This document contains the following:

- [Adding Your Oracle Cloud Services](#)
- [Getting up and Running with Your Java Cloud Service](#)
- [Validating with the Whitelist Scan](#)
- [Deploying to the Cloud](#)
- [Oracle Developer Cloud Service](#)

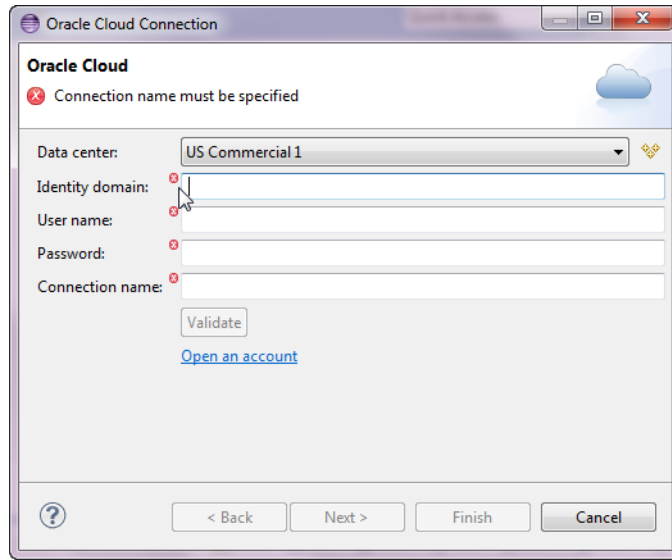
5.1 Adding Your Oracle Cloud Services

To add an Oracle Cloud Service to your application use the Cloud view, from the Java EE perspective.

To add a Cloud Service:

1. In the OEPE application Cloud view, choose **Connect**. The Oracle Cloud dialog appears, as shown in [Figure 5-1](#).
2. Enter the details in the dialog fields. When you sign up for an Oracle Cloud Service, the configuration details are emailed to you.
3. Click **Test Connection**. You can also test with **Perform whitelist scan prior to publish**. This does a local test to determine if your application deploys successfully to Oracle Cloud.
4. Click **Finish**.

Figure 5–1 Oracle Cloud Connection - Add Details



5.1.1 Using the Cloud View

Add and manage your cloud services using the Cloud view. Right clicking from the Cloud node gives you options to open your browser-based cloud portals, as well as Cloud connection properties, as shown in [Figure 5–2](#) and [Figure 5–3](#).

Right-clicking from your service node gives you access to a variety of features including Service Instances, Service Jobs, and the Service web-based console, as shown in [Figure 5–4](#). For more information see [Viewing the Java Cloud Service Jobs Log](#), and [Viewing the Java Cloud Service Instance Log](#).

Figure 5–2 Cloud View - Right-Click Server Instance Options

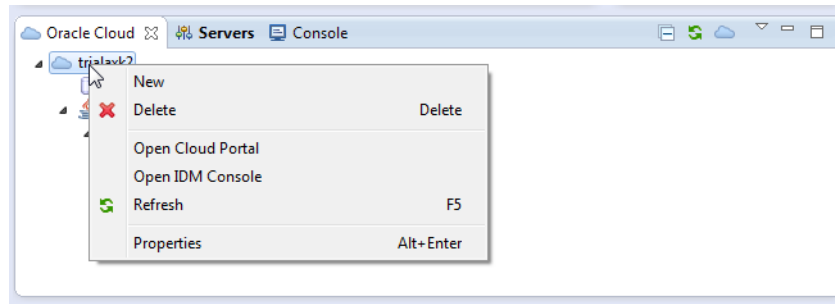
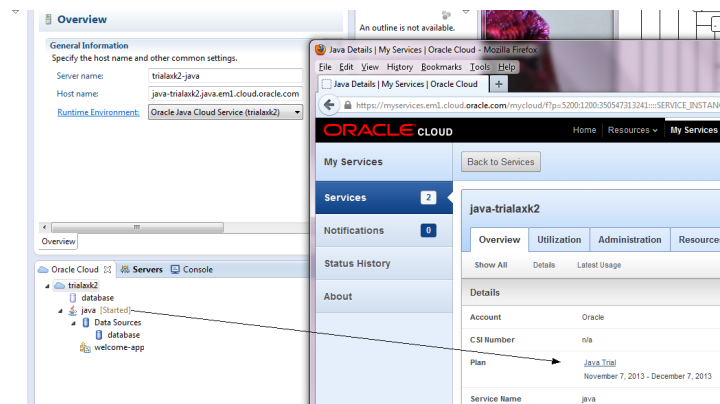
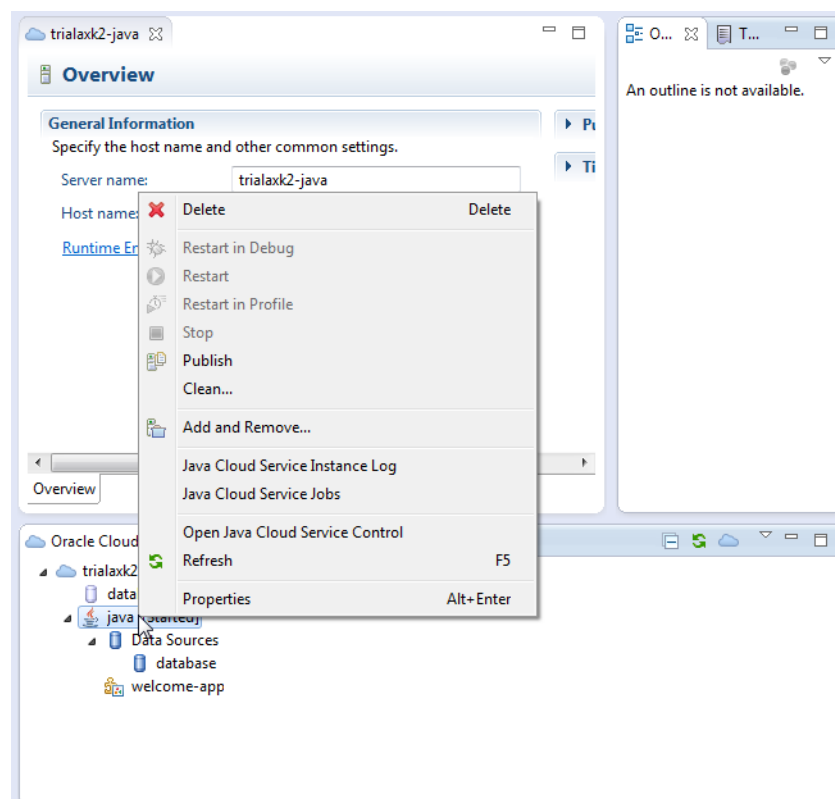


Figure 5–3 Cloud View And Corresponding Cloud Service Portal**Figure 5–4 Cloud View - Right-Click Java Service Options**

5.2 Getting up and Running with Your Java Cloud Service

With the Oracle Java Cloud Service you can manage the backend infrastructure of your OEPE applications without exposing the runtime to its service users. When you sign up for the Java Oracle Cloud Service, you get a deployment target for your OEPE application using a set of Java EE release 5, Java EE release 6, and Oracle WebLogic Server capabilities, in addition to a My Services web-based interface to manage your cloud tools.

For more information on the Java Cloud Service see, <http://cloud.oracle.com>.

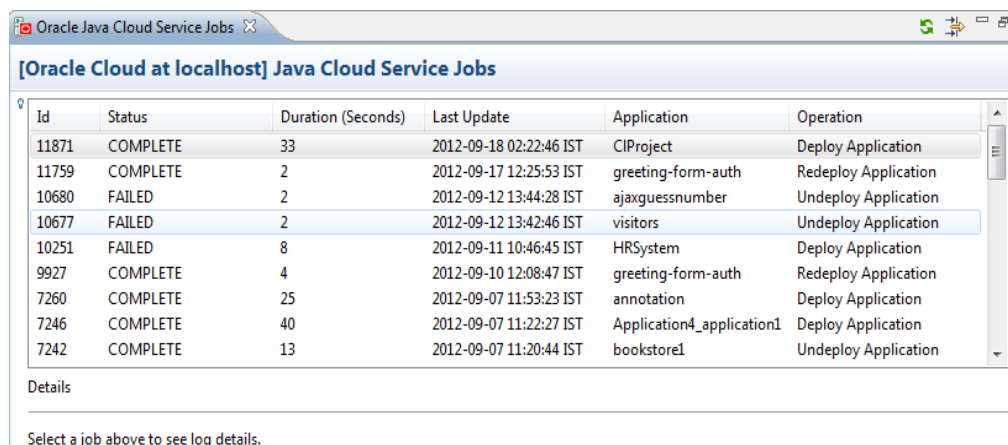
To add Java Cloud Service to your OEPE application:

1. Subscribe to the Java Cloud Service at <http://cloud.oracle.com>. You will get an email with the service details, which you will need to configure your OEPE application for the cloud.
2. Open your application.
3. Follow the steps in [Adding Your Oracle Cloud Services](#).

5.2.1 Viewing the Java Cloud Service Jobs Log

The Oracle Java Cloud Service Jobs log, shown in [Figure 5–5](#), provides comprehensive information on the jobs being executed by the Java Cloud Service, including Id, status, duration, and nature of your operation.

Figure 5–5 Oracle Java Cloud Service - Jobs Window



Id	Status	Duration (Seconds)	Last Update	Application	Operation
11871	COMPLETE	33	2012-09-18 02:22:46 IST	CIPProject	Deploy Application
11759	COMPLETE	2	2012-09-17 12:25:53 IST	greeting-form-auth	Redeploy Application
10680	FAILED	2	2012-09-12 13:44:28 IST	ajaxguessnumber	Undeploy Application
10677	FAILED	2	2012-09-12 13:42:46 IST	visitors	Undeploy Application
10251	FAILED	8	2012-09-11 10:46:45 IST	HRSsystem	Deploy Application
9927	COMPLETE	4	2012-09-10 12:08:47 IST	greeting-form-auth	Redeploy Application
7260	COMPLETE	25	2012-09-07 11:53:23 IST	annotation	Deploy Application
7246	COMPLETE	40	2012-09-07 11:22:27 IST	Application4_application1	Deploy Application
7242	COMPLETE	13	2012-09-07 11:20:44 IST	bookstore1	Undeploy Application

Details

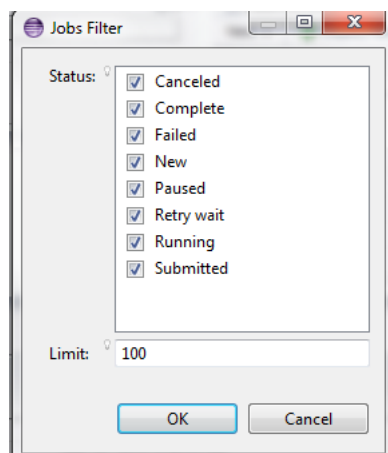
Select a job above to see log details.

To view the Oracle Java Cloud Service - Jobs log:

Right-click your service in the Servers view and select **Java Cloud Service Instance Log**.

You can filter jobs shown in the window by clicking the Filter icon in the toolbar to open the Jobs Filter, shown in [Figure 5–6](#).

Figure 5–6 Jobs Filter Dialog



5.2.2 Viewing the Java Cloud Service Instance Log

The Oracle Java Cloud Service Instance log provides detail on the performance of your services, as shown in [Figure 5-7](#).

Figure 5-7 Oracle Java Cloud Service Instance Log

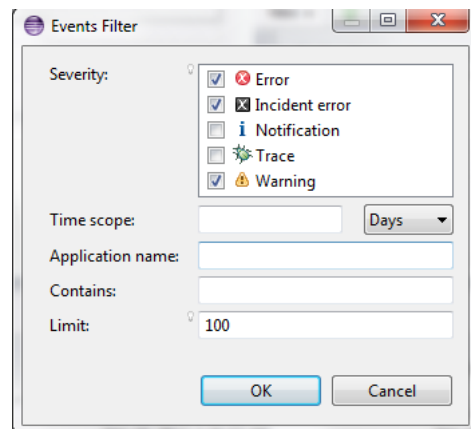
Message	Time
[ServletContext@351568230[app:secADF module:secured path:/secured spec-version:2.5]] Root cause of Ser...	2012-09-17 14:35:15 IST
ADF_FACES-60098:Faces lifecycle receives unhandled exceptions in phase RENDER_RESPONSE 6	2012-09-17 14:35:15 IST
[ServletContext@351568230[app:secADF module:secured path:/secured spec-version:2.5]] Root cause of Ser...	2012-09-17 14:33:23 IST
ADF_FACES-60098:Faces lifecycle receives unhandled exceptions in phase RENDER_RESPONSE 6	2012-09-17 14:33:23 IST
An attempt was made to execute the 'deploy' operation on an application named 'HRSystem' that is not cu...	2012-09-11 10:47:14 IST
An attempt was made to execute the 'deploy' operation on an application named 'HRSystem' that is not cu...	2012-09-11 10:47:12 IST
Failure occurred in the execution of deployment request with ID '1346996856071' for task '26'. Error is: 'webl...	2012-09-07 11:17:59 IST
User defined listener com.sun.faces.config.ConfigureListener failed: java.lang.RuntimeException: com.sun.f...	2012-09-07 11:17:58 IST
Failure occurred in the execution of deployment request with ID '1346996844117' for task '24'. Error is: 'webl...	2012-09-07 11:17:33 IST
User defined listener com.sun.faces.config.ConfigureListener failed: java.lang.RuntimeException: com.sun.f...	2012-09-07 11:17:33 IST
Failure occurred in the execution of deployment request with ID '1346987371463' for task '22'. Error is: 'webl...	2012-09-07 08:39:49 IST
User defined listener com.sun.faces.config.ConfigureListener failed: java.lang.RuntimeException: com.sun.f...	2012-09-07 08:39:49 IST
Failure occurred in the execution of deployment request with ID '1346987363341' for task '21'. Error is: 'webl...	2012-09-07 08:39:31 IST
User defined listener com.sun.faces.config.ConfigureListener failed: java.lang.RuntimeException: com.sun.f...	2012-09-07 08:39:30 IST
ADF_FACES-60096:Server Exception during PPR, #1	2012-09-07 08:37:35 IST
ADF_FACES-60098:Faces lifecycle receives unhandled exceptions in phase APPLY_REQUEST_VALUES 2	2012-09-07 08:37:35 IST
ADF_FACES-60098:Faces lifecycle receives unhandled exceptions in phase RENDER_RESPONSE 6	2012-09-06 07:54:24 IST
2012-09-05 18:49:53.726--ServerSession(367735080)--Exception [EclipseLink-4002] (Eclipse Persistence Servic...	2012-09-06 00:19:53 IST
2012-09-05 18:49:53.713--ServerSession(367735080)--Exception [EclipseLink-4002] (Eclipse Persistence Servic...	2012-09-06 00:19:53 IST
2012-09-05 18:49:53.705--ServerSession(367735080)--Exception [EclipseLink-4002] (Eclipse Persistence Servic...	2012-09-06 00:19:53 IST
2012-09-05 18:49:53.685--ServerSession(367735080)--Exception [EclipseLink-4002] (Eclipse Persistence Servic...	2012-09-06 00:19:53 IST
2012-09-05 18:49:53.68--ServerSession(367735080)--Exception [EclipseLink-4002] (Eclipse Persistence Servic...	2012-09-06 00:19:53 IST
2012-09-05 18:49:53.673--ServerSession(367735080)--Exception [EclipseLink-4002] (Eclipse Persistence Servic...	2012-09-06 00:19:53 IST
2012-09-05 18:49:53.666--ServerSession(367735080)--Exception [EclipseLink-4002] (Eclipse Persistence Servic...	2012-09-06 00:19:53 IST
2012-09-05 18:49:53.66--ServerSession(367735080)--Exception [EclipseLink-4002] (Eclipse Persistence Servic...	2012-09-06 00:19:53 IST

To view the Oracle Java Cloud Service Jobs log:

Right-click your service in the Servers view and select **Java Cloud Service Jobs**.

You can select the Filter icon in the toolbar to filter service instances using the Events Filter dialog, shown in [Figure 5-8](#).

Figure 5-8 Events Filter - Service Instances



5.3 Validating with the Whitelist Scan

The whitelist scan feature enables you to locally ensure that your application is valid for deployment to Oracle Cloud. Whitelist scanning can be invoked in three ways:

- **As you Type whitelist scan:** Occurs on source files that are currently open in the workspace, for example, Java class, or JSF/JSP pages.
- **Whitelist Builder:** When you select **File > Save**, the Whitelist Builder validates all files in the project, and reports violations in the Markers View, Problems View, and Whitelist Problems View.
- **On Demand whitelist scan:** To perform an on demand whitelist scan, right-click the project in the Project Explorer and select **Oracle Cloud Whitelist Scan**. An On Demand whitelist scan adds an additional level of scanning to everything on the project build path, for example, jar files. This type of scan is also automatically performed before deployment. Because it is a more expansive operation, it does not occur automatically during file operations like edit, save, or build.

5.4 Deploying to the Cloud

OEPE enables you to deploy your project to Oracle Cloud or Oracle WLS, depending on your preference.

To deploy to Oracle Cloud, use one of the following methods:

- Right-click the project you want to deploy, and select **Run As > Run on Server**. Then select **Oracle Cloud** on the Run on Server page and click **Finish**.
- Right-click the Oracle Cloud configuration in the Servers view and select **Add and Remove** to add the project you want to deploy to Oracle Cloud.

5.5 Oracle Developer Cloud Service

OEPE includes integration for Oracle Developer Cloud Service, which exposes the most common development tasks from the cloud directly from within the application. Oracle Developer Cloud Service is a collection of software and services hosted on the Oracle Cloud. It is a cloud-based software development Platform as a Service (PaaS) and a hosted environment for your application development infrastructure. It provides open-source, standards-based solutions to develop, collaborate, and deploy applications within Oracle Cloud.

Oracle Developer Cloud Service integration with OEPE includes the following:

- A dedicated Oracle Cloud view that displays Oracle Developer Cloud Service projects that you are a member of.
- Integration with Eclipse Mylyn and Oracle Developer Cloud Service Tasks system.
- Source Control System integration with Oracle Developer Cloud Service GIT repository.

5.5.1 Logging In to Oracle Developer Cloud Service

You can log in to Oracle Developer Cloud Service from Oracle Cloud view.

To open Oracle Cloud view:

1. Click **Window > Show View > Oracle Cloud**.

If the Oracle Cloud option is not available in the Show View submenu, click **Other** and choose **Oracle Cloud > Oracle Cloud** in the Show View dialog.

2. Click **Connect**.

Tip: The Oracle Cloud view is available in Java EE perspective, by default. To open the Java EE perspective, select **Window > Open Perspective > Java EE**.

If you are connecting to Oracle Developer Cloud Service for the first time, click the Connect link. In the Oracle Cloud Service Connection dialog, enter the following:

- **Data Center:** Select the Oracle Cloud data center.
- **Identity Domain:** Enter the identity domain of Oracle Developer Cloud Service.
- **Username and Password:** Enter the user name and password.
- **Connection Name:** Enter a name for the service instance, if required. By default, the connection name is set to the identity domain name.

After validating credentials, you are logged in to Oracle Developer Cloud Service from OEPE. After you log in, Oracle Cloud view displays all projects that are assigned to you.

5.5.2 Getting Up and Running with Your Developer Cloud Service

To log in to Oracle Developer Cloud Service from OEPE, open the Oracle Cloud view and click **New Cloud Connection Wizard**. The Oracle Cloud view is available in Java EE perspective. To open Java EE perspective, select **Window > Open Perspective > Java EE**.

To add Developer Cloud Service to your OEPE application:

1. Subscribe to the Developer Cloud Service at <http://cloud.oracle.com>. You will get an email with the service details, which you will need to configure your OEPE application for your cloud service.
2. Open your application.
3. Follow the steps in [Section 5.1, "Adding Your Oracle Cloud Services"](#). Choose the Developer Cloud Service from the server list.

5.5.3 Using the Oracle Developer Services Cloud View

Oracle Cloud View displays all your projects, and links to the common features of your service.

You can have multiple Oracle Developer Cloud Service accounts in the Oracle Cloud view. To add another account, select **Oracle Public Cloud** in Oracle Cloud view, right-click, and select **New**. Select the data center, enter the identity domain, user name and credentials, and click **Finish**.

Oracle Cloud view enables you to run the following actions for each project:

- Import projects from Oracle Developer Cloud Service, and export projects to Oracle Developer Cloud Service (see [Section 5.5.6, "Using eGit for DCS Source Control and Versioning"](#))
- Search and manage tasks (see [Section 5.5.11, "Updating Tasks"](#))
- Monitor builds of your project (see [Section 5.5.12, "Monitoring Hudson Builds"](#))

5.5.4 Importing an Oracle Developer Cloud Service Project

Importing a project from Oracle Developer Cloud Service to OEPE Workspace creates a local clone of Oracle Developer Service Git repository. After creating the local Git clone, Eclipse projects can be imported into the workspace.

To import a Project from Oracle Developer Cloud Service to OEPE Workspace:

1. In the Oracle Cloud view, expand the project, and select the Git repository from the Source node.
2. Double-click the selected Git repository to clone it to local machine.

A dialog appears informing that Git repository is ready for cloning. Click **OK**.

3. The cloned repository will be visible in Git Repositories view. If the view is not visible, open it from **Window > Show View > Git Repositories**.
4. Right-click the repository name in the Git Repositories view, and choose **Import Projects**.
5. Select a wizard to use for importing projects page of Import Projects from Git Repository dialog, choose the desired option, and click **Next**.

If you are not sure which option to select, select the **Import as General Project** option.

6. In the Import Project page of Import Projects from Git Repository dialog, verify the project name and directory, and click **Finish**.

5.5.5 Exporting a Project from OEPE to Oracle Developer Cloud Service

Exporting a project from OEPE workspace to Oracle Developer Cloud Service is useful if you already have an active local Git repository and want to push the source to the hosted Git repository.

To export a Project from OEPE Workspace to Oracle Developer Cloud Service:

1. Select the project in Project Explorer, right-click, and choose **Team > Share Project**.
2. In the Share Project wizard, select Git as the repository type, and click **Next**.
3. In the Configure Git Repository page, select the remote Git repository you would want to export the project to, and click **Finish**.
4. In Project Explorer, select the project, right-click, and choose **Team > Commit**.
5. In the Commit Changes dialog, enter commit description, select the files you want to commit, and click **Commit and Push**.
6. In the Push Results dialog, click **OK** to export the project to Oracle Developer Cloud Service.

5.5.6 Using eGit for DCS Source Control and Versioning

OEPE uses eGit for source control and version on DCS projects. EGit is an Eclipse Team provider for Git. Git is a distributed SCM, which means every developer has a full copy of all history of every revision of the code, making queries against the history very fast and versatile. For more information in eGit, see <http://www.eclipse.org/egit/>.

To open a Git repository view in OEPE, select **Window > Show View > Git Repositories**.

In some cases, the Git option may be under **other...**

On the right side of the Git view is a toolbar with the icons that you click to manage your Git features. Icon actions include:

- Collapse all
- Add an existing local repository
- Clone a Git repository
- Create a new Git repository
- Refresh your repository
- Adjust layout preferences
- Display latest branch commits

5.5.7 Using Git Tools in OEPE

Using the Team context menu in Project Explorer, you can perform various Git actions such as commit a project to Git repository, commit file changes, merge, branch, and push changes to the hosted Git repository.

To add a project to Local Git Clone:

1. Right-click the project, and select **Team > Share Project**.
2. In the Share Project wizard, select **Git** from the list of available source control systems, and click **Next**.
3. In Configure Git Repository page, select the Git repository, and click **Finish**.

5.5.8 Committing Changes to Oracle Developer Cloud Service Git Repository

Committing changes to the Oracle Developer Cloud Service Git repository is a two-step process: committing to the local Git clone or branch, and then pushing the changes to the cloud repository

To commit changes to Oracle Developer Cloud Service Git Repository:

1. To commit a change, right click the file, directory, or project and select **Team > Commit**.
2. In the Commit Changes dialog, add a commit message and select the files to be committed to the local repository, and click **Commit**.

Notes:

- If you have an Active Task, the Task ID and URL are automatically added to your commit message. For more information, see [Associating a Task with a Commit Transaction](#).
 - All committed changes appear in the home page of Browse module and the Activity Feed of Oracle Developer Cloud Service web interface.
-
-

5.5.9 Pushing Changes From the Local Git Repository to Oracle Developer Cloud Service Git Repository

To push latest updates from the local cloned Git repository to the Git cloud repository, use the Push command.

To push changes to Oracle Developer Cloud Service Git Repository

1. Open the Git Repositories view.
2. Right-click the repository, and then select **Push Upstream**.
3. In Push Results dialog, verify the message details, and then click **OK**.

5.5.10 Managing Documentation

You can also add wiki pages to your project from OEPE. Select and expand the project in Oracle Cloud view, select **Documentation**, right-click, select **Open in Browser**. The Wiki module home page opens in the web browser

From the Wiki home page, you can add or edit wiki pages of the project. If you have administrative privileges of a project, you can change the wiki markup type.

For more information about managing documentation and wiki pages, see [Section 5.5.10, "Managing Documentation"](#).

5.5.11 Updating Tasks

By default, following task queries are available in the Tasks node of Oracle Cloud view:

- All: Lists all tasks of the project
- Mine: Lists all tasks assigned to you
- Open: Lists all open tasks
- Recent: Lists all recently changed tasks
- Related: Lists all tasks related to you

Double-click a query to run it, and then double-click a task to open it in the Task Editor.

You can also perform the following tasks related actions from OEPE:

- Create a new query and import tasks, features, and defects from Oracle Developer Cloud Service
- Update imported tasks from OEPE
- Create local tasks and export them to Oracle Developer Cloud Service

5.5.11.1 Importing Tasks from Oracle Developer Cloud Service With a Custom Query

To import tasks from Oracle Developer Cloud Service, you would need to create a search query. For example, you can create a query to import all open defects assigned to you.

To import tasks with a custom query:

1. Select Tasks node of your project in Oracle Cloud view.
2. Right-click, and choose **New Query**.

3. In the Oracle Developer Cloud Service Tasks Query dialog, enter the search criteria.

For example, if you want to import all open defects assigned to you, enter the name of the query, select your name from the **Person** list, select **Defect** as Type, and **Assigned** as Status.

4. Click **Finish**.

All tasks matching the specified criteria will get imported to Task List view. To open Task List view, select **Window > Show View > Task List**. You can create multiple queries as per your requirement.

5.5.11.2 Creating a Local Task

To create a task, click the **New Task** icon in the Task List toolbar. When you create a Task, you can choose the repository as a local repository, or your project's repository on Oracle Developer Cloud Service.

If you choose Oracle Developer Cloud Service as the repository, the task will be automatically added to Oracle Developer Cloud Service and reflected in Tasks home page of the web interface.

5.5.11.3 Editing a Task

To view and edit a specific task, double-click the task in the Task List view to load it in the Task Editor. Update the task, and click **Submit**. If it is a local task, it will be updated in local task repository. If it is a Oracle Developer Cloud Service task, it will be automatically updated in Oracle Developer Cloud Service task repository.

5.5.11.4 Synchronizing Tasks with Oracle Developer Cloud Service

To synchronize a task (or a query) between OEPE and Oracle Developer Cloud Service, right-click the task (or the query) in Task List view, and choose **Synchronize**.

5.5.11.5 Associating a Task with a Commit Transaction

If you want to associate a task with a commit transaction, you should activate the task first. To activate a task, select and right-click the task in Task List view, and choose **Activate**.

OEPE also provides integration with Mylyn. Activating a task enables Mylyn to track which files are related to the current task. Mylyn automatically hides files that are not related to the active task. When committing changes to Git for the Active task, your commit message automatically references the Task ID in Oracle Developer Cloud Service. This effectively creates a link between the code commit and the task allowing for easy traceability. Links between source commits and Tasks are also reflected in the web interface of Oracle Developer Cloud Service.

You can find more information about working with Mylyn Tasks and the Tasks user interface in the Mylyn User Guide at the following URL:

http://wiki.eclipse.org/Mylyn_User_Guide

5.5.12 Monitoring Hudson Builds

To view all jobs of the Hudson builds, expand the Build node of a project in Oracle Cloud view. To perform any action, select the job (or the build) and click **Open in Browser**. The Builds module of Oracle Developer Cloud Service opens in the web browser.

Maven Support

Maven is an open source build management tool that is central to project build tasks such as compilation, packaging, and artifact management. Maven uses a strict XML-based ruleset to promote consistency while maintaining flexibility. Most Java-centric continuous integration systems integrate well with Maven, making it a good choice for an underlying build system. This chapter describes how use Maven for Oracle Enterprise Pack for Eclipse (OEPE).

This document contains the following sections:

- [Using Maven with OEPE](#)
- [Setting up Your Maven Environment](#)
- [Creating a Maven Settings File](#)
- [Populating the Maven Repository](#)
- [Installing the Maven Archetypes](#)
- [Creating ADF Applications with Maven Integration](#)
- [Importing Maven Projects](#)
- [Using Maven to Deploy to a WebLogic Server](#)

6.1 Using Maven with OEPE

Maven is installed with your Fusion Middleware in your Oracle home at `ORACLE_HOME/oracle_common/modules/org.apache.maven_3.0.4`. This is a copy of the standard Maven 3.0.4 release, without any modifications. You can also get Maven from the Apache website.

To use Maven as the build tool for developing ADF applications in OEPE, you need to populate the Maven repository with Oracle artifacts. For populating the repository, a Maven Synchronization plug-in is provided, which allows you to populate a local or shared Maven repository from an Oracle home. When you install a Fusion Middleware 12c product, the Maven archetypes, plug-ins, and Project Object Models (POMs) are installed with the product where the synchronization plug-in can find them.

Using Maven with OEPE requires you to configure and setup the maven environment using the following steps:

1. [How to Set Up Your Maven Environment](#)
2. [How to Create Your Maven Settings File](#)
3. [How to Use the Oracle Maven Synchronization Plug-In](#)
4. [How to Install the Maven Archetypes](#)

For further information see *Developing Applications Using Continuous Integration*.

Note: After you patch your Oracle home, you should run the plug-in again to ensure that your Maven repository matches Oracle home. This ensures that your builds use correct versions of all artifacts in that particular environment. For information on patching, see Chapter 5, "Installing and Configuring Maven for Build Automation and Dependency Management" in *Developing Applications Using Continuous Integration*.

6.2 Setting up Your Maven Environment

Once you have Maven installed, either as part of a Fusion Middleware package, or from the Apache website, you need to perform a few initial steps to set up your environment.

6.2.1 How to Set Up Your Maven Environment

The first thing you need to do to use Maven is to set your environment variables.

To set up your environment on Unix:

1. Add Maven to your operating system's PATH:
 - Update your shell startup script, your `.profile` or `.bash_profile`, to update the path. For example, if you have installed Oracle WebLogic Server in `/u01/fmwhome` and you are using the bash shell, then add the following to the PATH environment variable:

```
export M2_HOME=/u01/fmwhome/oracle_common/modules/org.apache.maven_
3.0.4
export PATH=${M2_HOME}/bin:$PATH
```

2. Set the JAVA_HOME environment variable to point to your JDK installation, for example, `export JAVA_HOME=u01\jdk1.7.0`

To set up your environment on Windows:

1. Edit your PATH environment variable and add the correct Maven directory path at the beginning of the PATH environment variable. For example, if you have installed your Fusion Middleware in `C:\fmwhome`, then add the following:

```
C:\fmwhome\oracle_common\modules\org.apache.maven_3.0.4\bin
```

2. Set the JAVA_HOME environment variable to point to your JDK installation, for example, `export JAVA_HOME=C:\Oracle\Middleware\jdk1.7.0`
3. Add the M2_HOME environment variable by opening up the system properties (WinKey + Pause), selecting the Advanced tab, then the Environment Variables button. Add the M2_HOME variable in the user variables with the value `C:\<ORACLE_HOME>\oracle_common\modules\org.apache.maven_3.0.4\bin\.` If you installed Maven from the Apache website, the value will be `C:\Program Files\Apache Software Foundation\apache-maven-3.0.5.`
4. In the same dialog, add the user variable with the value `%M2_HOME%\bin.`
5. Open a new command prompt (Winkey + R then type `cmd`) and run `mvn --version` to verify that it is correctly installed.

6. If you have any issues with this process, you can go to this Apache page to problem solve:
<https://cwiki.apache.org/confluence/display/MAVEN/NoPluginFoundForPrefixException>.

6.3 Creating a Maven Settings File

To use Maven you will need a settings file, if any of the following are true:

- You have installed Maven for the first time.
- You are working behind a firewall or proxy server.
- Your organization has its own internal Maven Repository Manager.

6.3.1 How to Create Your Maven Settings File

To create a Maven settings file, create a XML file called `settings.xml`. If your user name is Bob, then the directory path for the `settings.xml` file should be the following:

```
/home/bob/.m2/settings.xml.
```

For Windows, If your user name is Bob then the directory path for the `settings.xml` file should be the following:

```
/users/bob/.m2/settings.xml.
```

You are going to add the following information to your settings file:

- **Proxy** - sets the HTTP proxy server that is required to access Maven repositories on the Internet.
- **Servers** - sets the credentials for the Maven repository, so that you do not have to enter them every time you want to access the repository.
- **Mirrors** - directs that instead of trying to access the Maven central repository directly, it should use your internal Maven repository manager as a mirror (cache) of Maven central repository.

Add these settings as shown in [Example 6-1](#).

Example 6-1 Maven Settings File

```
<settings>
<proxies>
<proxy>
<active>true</active>
<protocol>http</protocol>
<host>proxy.mycompany.com</host>
<port>8080</port>
<nonProxyHosts>mycompany.com</nonProxyHosts>
</proxy>
</proxies>
<servers>
<server>
<id>maven.mycompany.com</id>
<username>me@mycompany.com</username>
<password>{COQLCE6DU6GtcS5P=}</password>
</server>
</servers>
<mirrors>
<mirror>
```

```
<id>archiva</id>
  <name>Internal Archiva Mirror of Central</name>
  <url>http://archiva.mycompany.com/repositories/internal</url>
  <mirrorOf>central</mirrorOf>
</mirror>
</mirrors>
</settings>
```

For more information about available Maven settings, see the Apache Maven documentation at, <http://maven.apache.org/settings.html>

6.4 Populating the Maven Repository

The next step is populating your Maven repository with the Oracle artifacts. The Oracle Maven Synchronization plug-in populates a local or shared Maven repository from an Oracle home. When you run the plug-in, it finds the Oracle artifacts and transfers them to a Maven repository.

6.4.1 How to Use the Oracle Maven Synchronization Plug-In

To use the plug-in start by specifying the location of the Oracle home and the location of the Maven repository using either a file system path, or a URL. The plug-in checks for all Maven artifacts in the Oracle home, then ensures that the artifacts are installed in the specified Maven repository, and that the versions match. This ensures the version numbers and the files match at the binary level, and that all patched files are reflected accurately in the Maven repository.

The plug-in is located in your ORACLE_HOME directory path and consists of two files:

- The Maven POM file that describes the plug-in, which is located at ORACLE_HOME/oracle_common/plugins/maven/com/oracle/maven/oracle-maven-sync/12.1.3/oracle-maven-sync.12.1.3.pom
- The JAR file that contains the plug-in, which is located at ORACLE_HOME/oracle_common/plugins/maven/com/oracle/maven/oracle-maven-sync/12.1.3/oracle-maven-sync.12.1.3.jar

To run the plug-in you first need to install it on your Maven repository. You can install it into your local repository on your computer, or you can deploy it into a remote shared internal repository.

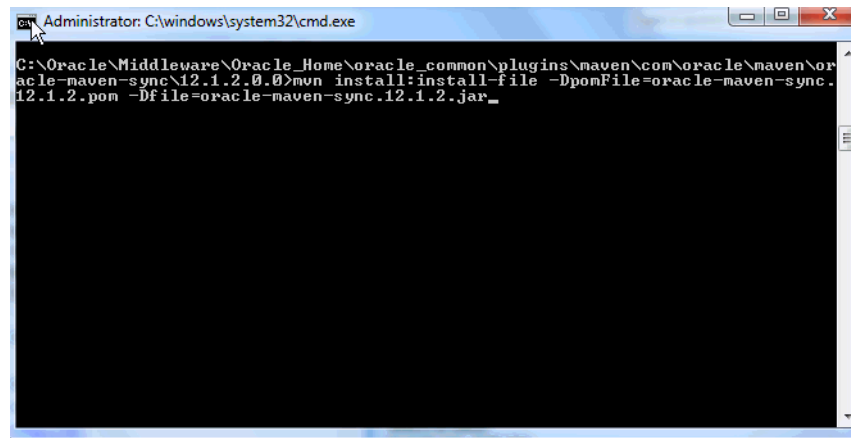
To install the plug-in to a local repository using the command line:

1. In a command prompt, navigate to the location of your Maven plug-in jar located at ORACLE_COMMON/oracle_common/plugins/maven/com/oracle/maven/oracle-maven-sync/12.1.3
2. Run the following command to install to the default repository:

```
mvn install:install-file -DpomFile=oracle-maven-sync.12.1.3.pom
-Dfile=oracle-maven-sync.12.1.3.jar
```

Run the following command to install to a specified directory:

```
mvn install:install-file -DpomFile=oracle-maven-sync.12.1.3.pom
-Dfile=oracle-maven-sync.12.1.3.jar -Dmaven.repo.local=<path to local
repository>
```

Figure 6–1 Using the Oracle Maven Synchronization Plug-In Install Command

To install the plug-in to a shared internal repository using the command line:

1. In a command prompt, navigate to the location of your plug-in jar at ORACLE_COMMON/oracle_common/plugins/maven/com/oracle/maven/oracle-maven-sync/12.1.3
2. Run the following command:


```
mvn deploy:deploy-file
-DpomFile=oracle-maven-sync-12.1.3.pom
-Dfile=oracle-maven-sync-12.1.3.jar
-Durl=http://servername/archiva/repositories/internal
-DrepositoryId=internal
```

For information about the deploy command see the Maven documentation at, <http://maven.apache.org/plugins/maven-deploy-plugin/deploy-file-mojo.html>

6.4.2 Running the Oracle Maven Synchronization Plug-in

The Oracle Maven Synchronization plug-in supports two Maven goals:

- **Help** - prints out help information. Execute the help goal by running `mvn com.oracle.maven:oracle-maven-sync:help`
- **Push** - populates a repository. Goal semantics depend on how you define your plug-in parameters.

To populate your repository, run the plug-in using the push goal. The push goal requires you define your plug-in parameters. You can populate a local or a remote repository using the push goal. Push goal parameters are defined in Table 6–1. You can specify the parameters either in your command line or in your POM file.

Table 6–1 Push Goal Parameters and Descriptions

Features	Description
ServerID	A pointer to the server entry in your Maven settings.xml file. This is required only if you intend to deploy to a remote repository. The settings.xml should provide the remote artifact repository deployment information, such as URL, user name, and password.
oracleHome	The path to the Oracle home that you wish to populate the Maven repository from.

Table 6–1 (Cont.) Push Goal Parameters and Descriptions

Features	Description
testingOnly	This controls whether the plug-in attempts to publish the artifacts to the repository. If you test this to <code>true</code> , which is the default value, then the push goal will find all of your POM files and print out details of what it would have done if this is set to <code>false</code> . However, it does not publish any artifacts or make any change to the system.
failOnError	If you set this property to <code>false</code> and the plug-in fails to process a resource, it continues to process all resources. Failures are logged as warnings, but the process completes successfully. If you set this property to <code>true</code> , when it encounters the first problem the plug-in will immediately exit with an error. This is the default.

6.4.2.1 Populating a Local Repository

If you are populating a local repository, specify `oracleHome` and `testingOnly=false`.

The `localRepository` element in your `settings.xml` file indicates the location of your local Maven repository. If you exclude the `localRepository` element in your `settings.xml`, the default location is in the `${HOME}/.m2/repository` directory.

If you want to override the `localRepository` value, then you must specify the override location on the command line as a Maven option, for example,

```
mvn com.oracle.maven:oracle-maven-sync:push
-Doracle-maven-sync.oracleHome=/path/to/oracleHome
-Dmaven.repo.local=/alter/nate/path
```

To specify the parameters in your POM file, you must add a plug-in entry similar to the following in your POM file, as shown in [Table 6–2](#).

Example 6–2 Plug-in Parameters in Your POM File

```
<plugin>
<groupId>com.oracle.maven</groupId>
<artifactId>oracle-maven-sync</artifactId>
<version>12.1.3</version>
<configuration>/home/<name>/Oracle/Middleware</oracleHome>
</configuration>
</plugin>
```

To populate a local repository:

1. Open the command prompt and navigate to the following directory in your OEPE install: `{install}/plugins/oracle.eclipse.tools.adf/maven/ADF Basic Application Archetype`.
2. Run the following command if you HAVE NOT defined the parameters in your POM file:

```
mvn com.oracle.maven:oracle-maven-sync:push
-Doracle-maven-sync.oracleHome=/path/to/oracleHome-Doracle-maven-sync.t
estingOnly=false
```

Run the following command if you HAVE defined the plug-in parameters in your POM file:

```
mvn com.oracle.maven:oracle-maven-sync:push
```

6.4.2.2 Populating a Remote Repository

If you are populating a remote repository, specify `serverId` and `oracleHome` on the command line or in the plug-in configuration. You must also have a repository configuration in your `settings.xml` that matches the server Id you provide to the plug-in. If authentication is required for deployment, you must also add a server entry to your Maven `settings.xml`, for example:

```
mvn com.oracle.maven:oracle-maven-sync:push
-Doracle-maven-sync.oracleHome=/path/to/oracleHome
-Doracle-maven-sync.serverId=internal
```

In your `settings.xml` file, you must define the target repository in a profile, and activate that profile using the `activeProfiles` tag as shown in [Example 6-3](#).

Specify an encrypted password in the server section. For details on how to encrypt the server passwords, see the Apache website at http://maven.apache.org/guides/mini/guide-encryption.html#How_to_encrypt_server_passwords

Example 6-3 *Maven setting.xml with Authentication Details*

```
<profiles>
  <profile>
    <id>default</id>
    <repositories>
      <repository>
        <releases>
          <enabled>true</enabled>
          <updatePolicy>always</updatePolicy>
          <checksumPolicy>warn</checksumPolicy>
        </releases>
        <snapshots>
          <enabled>true</enabled>
          <updatePolicy>never</updatePolicy>
          <checksumPolicy>fail</checksumPolicy>
        </snapshots>
        <id>internal</id>
        <name>Team Internal Repository</name>
        <url>http://some.host/maven/repo/internal</url>
        <layout>default</layout>
      </repository>
    </repositories>
  </profile>
</profiles>
...
<server>
  <id>internal</id>
  <username>deployer</username>
  <password>welcome1</password>
  <password>welcome1</password>
</server>
...
<activeProfiles>
  <activeProfile>default</activeProfile>
</activeProfiles>
```

To specify the parameters in your POM, add a plug-in entry similar to [Example 6-4](#).

Example 6–4 POM File Parameters

```

<plugin>
<groupId>com.oracle.maven</groupId>
<artifactId>oracle-maven-sync</artifactId>
<version>12.1.3</version>
  <configuration>
    <serverId>internal</serverId>
<oracleHome>/path/to/oracleHome</oracleHome>
    <testOnly>>false</testOnly>
  </configuration>
</plugin>

```

To populate a remote repository:

1. Open the command prompt and navigate to the following directory in your Maven install: `{install}/plugins/oracle.eclipse.tools.adf/maven/ADF Basic Application Archetype`.

2. Run the following command:

```
mvn com.oracle.maven:oracle-maven-sync:push
```

6.4.2.3 What Happens When You Run a Push Goal to Populate a Repository?

When you run the push goal, the following actions are completed:

- Checks the Oracle home you have provided and makes a list of all of the Maven artifacts inside that Oracle home. This is done by looking for POM files in the `ORACLE_HOME/oracle_common/plugins/maven dependencies` directory and its subdirectories, recursively and in the `ORACLE_HOME/PRODUCT_HOME/plugins/maven` directory and its subdirectories recursively for each `PRODUCT_HOME` that exists in the `ORACLE_HOME`.
- Checks if the JAR file referred to by each POM file is available in the Oracle home.
- Calculates a SHA1 checksum for the JAR file.
- Attempts to publish the JAR, POM, and SHA1 files to the repository that you have provided.

The following types of Maven artifacts are installed into your repository:

- Maven dependencies provided by Oracle, which includes the following:
 - Client API classes
 - Compilation, packaging, and deployment utilities, for example, `appc` and `wlst`
 - Component JARs that must be embedded in the application
 - Client-side runtime classes, for example, `t3` and JAX-WS client runtimes
- Maven plug-ins provided by Oracle that handle compilation, packaging, and deployment
- Maven archetypes provided by Oracle that provide project templates

6.5 Installing the Maven Archetypes

To populate your repository with the Maven archetypes, run the clean install, and archetype update local catalog commands, as shown in [Figure 6–2](#) and [Figure 6–3](#).

6.5.1 How to Install the Maven Archetypes

Install the Maven archetypes from the command line.

To install the Maven archetypes:

1. Open a command prompt and navigate to the archetype directory of your plugins
install: {OEPE install}/plugins/oracle.eclipse.tools.adf/maven/ADF Basic Application Archetype
2. Run the following command:
mvn clean install
3. Run a second command:
mvn archetype:update-local-catalog

Figure 6–2 Maven Archetype Maven Clean Install Command

```

Administrator: C:\Windows\system32\cmd.exe
D:\penny\oepe_juno_builds\june 11\plugins\oracle.eclipse.tools.adf_6.2.0.201306100237\maven\ADF Basic Application Archetype>mvn clean install
[INFO] Downloaded: http://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-utils/3.0/plexus-utils-3.0.jar (221 KB at 55.2 KB/sec)
[INFO] Downloaded: http://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-io/2.0.1/plexus-io-2.0.1.jar (57 KB at 10.3 KB/sec)
[INFO] Building jar: D:\penny\oepe_juno_builds\june 11\plugins\oracle.eclipse.tools.adf_6.2.0.201306100237\maven\ADF Basic Application Archetype\target\adf-basic-application-12.1.2-0.jar
[INFO] --- maven-install-plugin:2.3.1:install (default-install) @ adf-basic-application ---
[INFO] Installing D:\penny\oepe_juno_builds\june 11\plugins\oracle.eclipse.tools.adf_6.2.0.201306100237\maven\ADF Basic Application Archetype\target\adf-basic-application-12.1.2-0.jar to C:\Users\peander.ST-USERS\.m2\repository\com\oracle\adf\archetypes\adf-basic-application\12.1.2-0\adf-basic-application-12.1.2-0.jar
[INFO] Installing D:\penny\oepe_juno_builds\june 11\plugins\oracle.eclipse.tools.adf_6.2.0.201306100237\maven\ADF Basic Application Archetype\pom.xml to C:\Users\peander.ST-USERS\.m2\repository\com\oracle\adf\archetypes\adf-basic-application\12.1.2-0\adf-basic-application-12.1.2-0.pom
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 1:02.132s
[INFO] Finished at: Tue Jun 11 15:18:42 PDT 2013
[INFO] Final Memory: 13M/147M
[INFO] -----
D:\penny\oepe_juno_builds\june 11\plugins\oracle.eclipse.tools.adf_6.2.0.201306100237\maven\ADF Basic Application Archetype>mvn clean install

```

Figure 6–3 Maven Archetype Update Local Catalog Command

```

Administrator: C:\Windows\system32\cmd.exe
D:\penny\oepe_juno_builds\june 11\plugins\oracle.eclipse.tools.adf_6.2.0.201306100237\maven\ADF Basic Application Archetype>mvn archetype:update-local-catalog
[INFO] Scanning for projects...
[INFO]
[INFO] Building Oracle ADF Basic Application Archetype 12.1.2-0
[INFO]
[INFO] --- maven-archetype-plugin:2.2:update-local-catalog (default-cli) @ adf-basic-application ---
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 1.480s
[INFO] Finished at: Tue Jun 11 15:28:17 PDT 2013
[INFO] Final Memory: 14M/215M
[INFO] -----
D:\penny\oepe_juno_builds\june 11\plugins\oracle.eclipse.tools.adf_6.2.0.201306100237\maven\ADF Basic Application Archetype>mvn archetype:update-local-catalog

```

6.6 Creating ADF Applications with Maven Integration

You can create an ADF application with Maven integration in the following ways:

- From a command-line using the OEPE provided ADF Application Archetype

- From OEPE using the New Maven Project and using the ADF application Archetype
- From OEPE using the New ADF Application Wizard

6.6.1 How to Create an ADF Application with Maven Integration from the Command Line

You can create an ADF application using the command line.

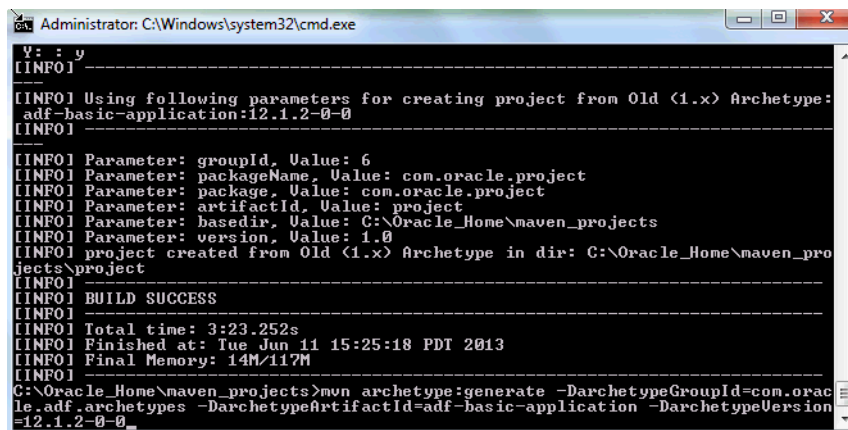
To create an ADF application from the command line:

1. Open a command prompt and navigate to your project location.
2. Run the following command:


```
mvn archetype:generate -DarchetypeGroupId=com.oracle.adf.archetype
-DarchetypeArtifactId=adf-basic-application
-DarchetypeVersion=12.1.3-0-0
```
3. Enter the appropriate required values for `groupId`, `artifactId`, `version`, and `package`, as shown in [Figure 6-4](#).
4. Change directory to newly-created top-level Maven project folder.
5. Execute the following command:

```
mvn clean install
```

Figure 6-4 Running the Generate Basic Maven Application Command



```
Administrator: C:\Windows\system32\cmd.exe
V: : y
[INFO] -----
[INFO] Using following parameters for creating project from Old (1.x) Archetype:
adf-basic-application:12.1.2-0-0
[INFO] -----
[INFO] Parameter: groupId, Value: 6
[INFO] Parameter: packageName, Value: com.oracle.project
[INFO] Parameter: package, Value: com.oracle.project
[INFO] Parameter: artifactId, Value: project
[INFO] Parameter: basedir, Value: C:\Oracle_Home\maven_projects
[INFO] Parameter: version, Value: 1.0
[INFO] project created from Old (1.x) Archetype in dir: C:\Oracle_Home\maven_projects\project
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 3:23.252s
[INFO] Finished at: Tue Jun 11 15:25:18 PDT 2013
[INFO] Final Memory: 14M/117M
[INFO] -----
C:\Oracle_Home\maven_projects>mvn archetype:generate -DarchetypeGroupId=com.oracle.adf.archetypes -DarchetypeArtifactId=adf-basic-application -DarchetypeVersion=12.1.2-0-0
```

6.6.2 How to Create an Maven Project with Maven integration from the Wizard

You can create a Maven application using the Maven Project wizard.

To create a Maven application from the application wizard:

1. In your OEPE Eclipse application go to **File > New > Project**. In the New Project dialog expand the Maven directory and select **Maven Project** and click **Next**.
2. In the New Maven Project dialog, select your project name and location, as shown in [Figure 6-5](#).
3. Click **Next**. The Select an Archetype dialog appears. Select the ADF archetype called `com.oracle.adf.archetypes`, as shown in [Figure 6-6](#). Type "adf" as a filter to filter out the many other options.

4. Select the ADF archetype and click **Next**.
5. Enter your values in the dialog fields, as shown in [Figure 6-7](#).
6. Click **Finish**. The new Maven application appears in your Project Explorer window.

Figure 6-5 Creating a New Maven Project - Filename and Location

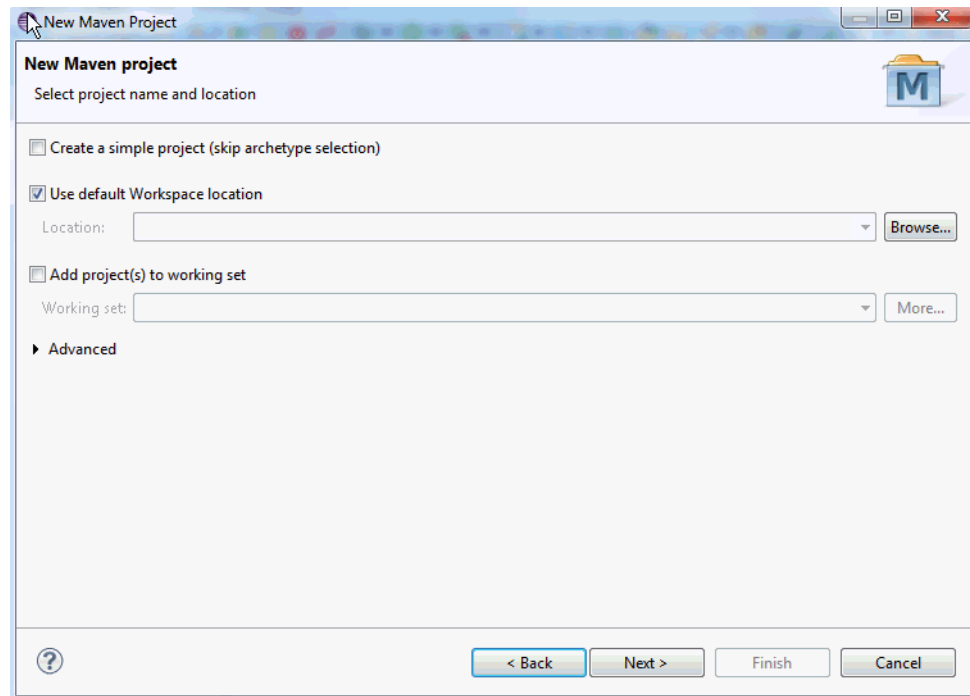


Figure 6-6 Creating a New Maven Project - Selecting the ADF Archetype

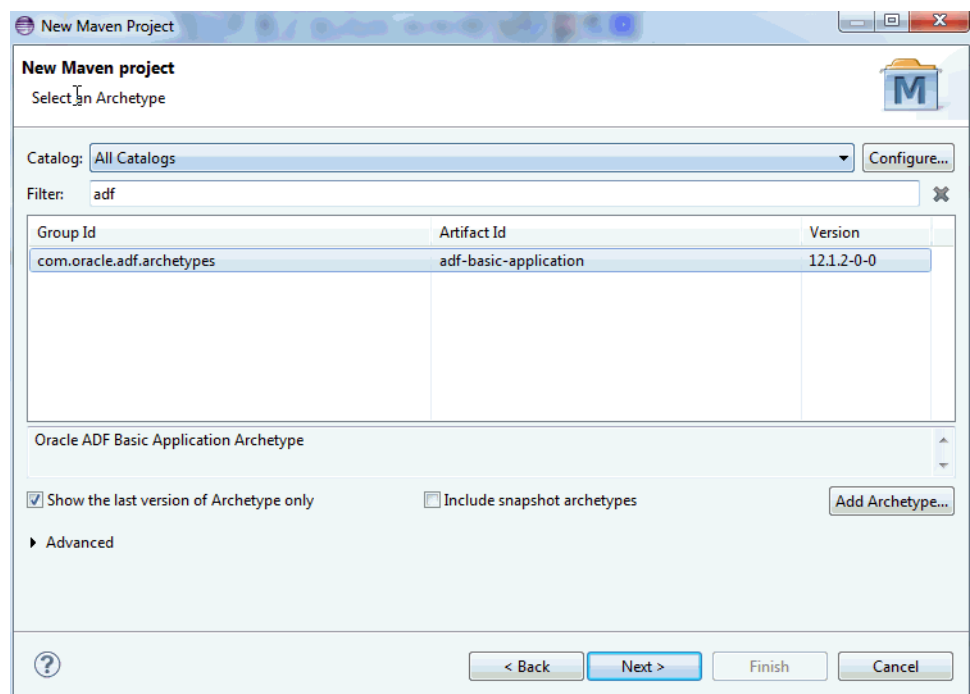
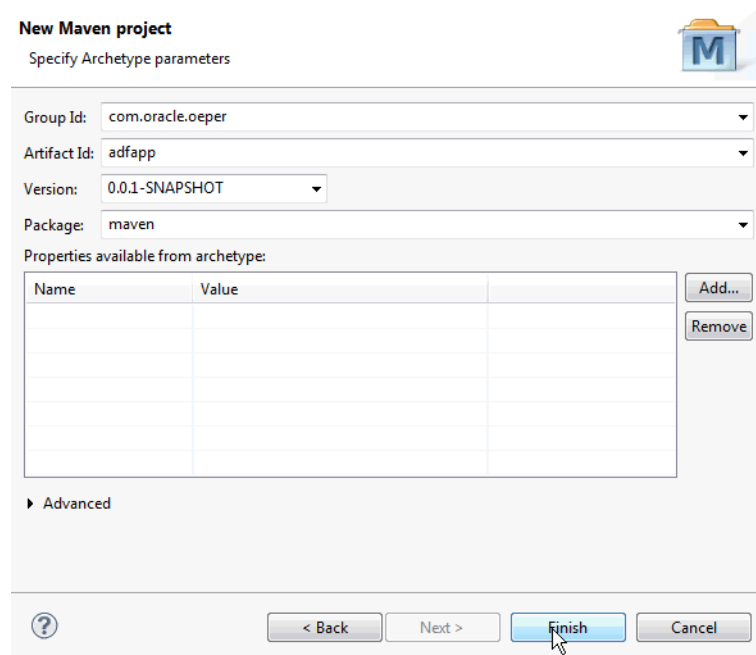


Figure 6–7 Creating a New Maven Project - Specifying the Archetype Parameters

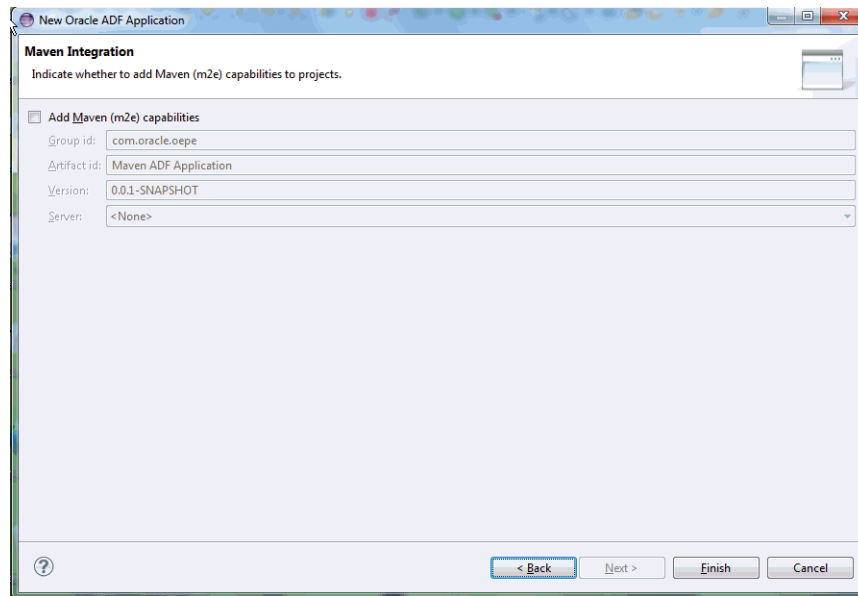


6.6.3 How to Add Maven Integration to New ADF Application Projects

OEPE provides the option to add Maven capabilities to projects in your new ADF application.

To add Maven integration to new ADF application projects:

1. Go to **File > New > ADF Application**. You are creating an ADF application in which you will add Maven capabilities to the application projects.
2. Add the information required for you new ADF application on the first two wizard pages.
3. The third wizard page is the Maven Integration Page, as shown in [Figure 6–8](#).
4. Check the **Add Maven (m2e) capabilities to projects** box. This adds Maven capabilities.
5. Click **Finish**.

Figure 6–8 Add Maven Integration Page in New ADF Application Wizard

6.7 Importing Maven Projects

In OEPE you can import your existing Maven projects into your working application. ADF applications can be imported as well as any J2EE Maven application built outside the OEPE framework.

To import an existing Maven project:

1. In your OEPE application choose **File > Import > Maven > Existing maven projects**.
2. Click **Next**. the Select Maven Projects dialog appears. Enter the root directory for the Maven project you are importing, and select your project, as shown in [Figure 6–9](#).
3. Click **Next**. The connectors dialog appears. Setup your Maven plugin connectors, as shown in [Figure 6–10](#).
4. Click **Finish**.

Figure 6–9 Importing Maven Projects _ Available Projects

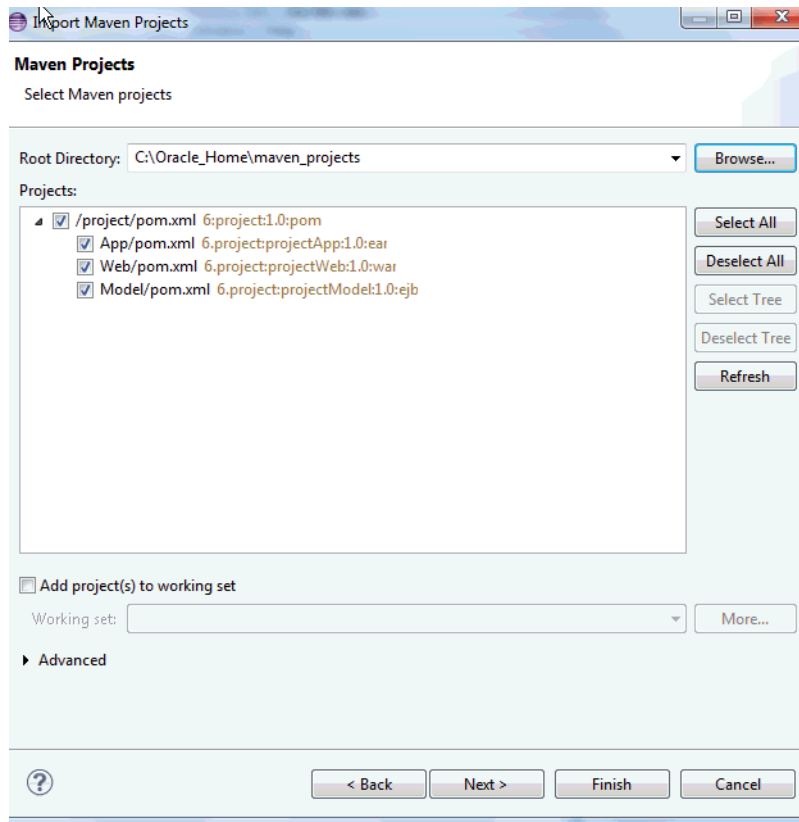
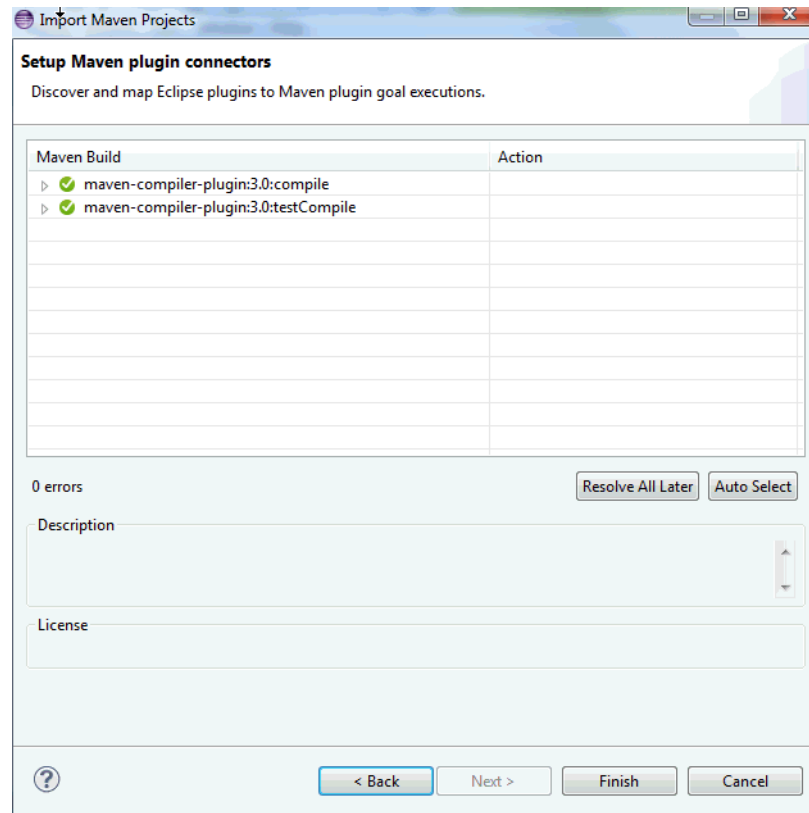


Figure 6–10 Importing Maven Projects - Connectors

6.8 Using Maven to Deploy to a WebLogic Server

Use the Maven plug-in to deploy, redeploy applications built using Maven to WebLogic Server from within the Maven environment.

6.8.1 How to Deploy using Maven to a Running WebLogic Server

Use the command line to deploy using Maven to a running WebLogic server instance.

To Use the Command Line to Deploy using Maven to a Running WebLogic Server:

1. Open a command prompt and navigate to `.../{artifactId}/App`, where `{artifactId}` is the name specified during archetype creation.
2. Run the following command:

```
mvn com.oracle.weblogic:weblogic-maven-plugin:redeploy
-Dwls.adminurl=t3://localhost:7001 -Dwls.user=weblogic
-Dwls.password=welcome1
```

Web Services Support

OEPE lets you to build enterprise-class Web services that employ standard Web service technologies, such as XML, SOAP, and WSDL. OEPE simplifies Web service development by allowing you to focus on application logic, rather than the complex implementation details traditionally required by these technologies.

This document contains the following sections:

- [Starting Points of Web Services Development with OEPE](#)
- [Creating Web Services Projects](#)
- [Generating Client Code for Web Services](#)
- [Generating JAXB Types](#)
- [Using Client Proxy Templates](#)
- [Using WebLogic Web Services Annotations View](#)
- [Validating Web Services Projects](#)
- [Generating Web Services for Spring Service Beans](#)
- [Configuring HTTPS Client Credentials](#)

7.1 Starting Points of Web Services Development with OEPE

This section describes how to get started with developing web services.

7.1.1 Generating a Web Service From a WSDL File

Using OEPE, you can generate a Web service from a WSDL file. The resulting Web service class will contain the public endpoint interface described by the WSDL without the implementation. After the Web service has been generated, you have to fill in the Web service implementation details.

To generate a Web service from a WSDL:

1. Create a new WSDL file or import an existing WSDL file into your project. For more information, see [Section 7.1.6, "Creating a New WSDL File"](#) and [Section 7.1.5, "Imported WSDL Files."](#)
2. In the Project Explorer, right-click the WSDL file and select **WebLogic Web Services > Generate Web Service**.

This opens the New Web Service from WSDL dialog that lets you specify many parameters for the generated Web service, including the port, the output locations, and Ant script generation options.

Selecting **Keep generated Ant Script** saves an Ant file for modification and reuse of the generation process.

In addition to the Web service implementation class, this will create a JAR file that contains a Web service interface class, as well as types referenced in the original WSDL file. The default location for the JAR file is the project's `WEB-INF/lib` directory. If you select a different location that is not on the class path, your Web service is unlikely to function properly.

You can test your Web service using the Test Client. For more information, see [Section 7.1.8, "Testing Web Services."](#)

You can also customize your Web service. For more information, see [Section 7.1.1.1, "Customizing a Web Service."](#)

7.1.1.1 Customizing a Web Service

You can customize your Web Service using the JAX-WS bindings file that points to an appropriate WSDL file.

A JAX-WS bindings file lets you change the shape of the Java Web service artifacts generated through WSDL. The bindings allow you to modify class and method names, method signatures (wrapper versus non-wrapper), add Javadoc, and attach handlers. Sources of information on the bindings file specifications with examples can be found in the references section.

To create a bindings file:

1. Right-click in the Project Explorer and select **New > Other** from the drop-down menu. This opens the New dialog that lets you select a wizard to use.
2. Select **Oracle > WebLogic > Web Services > JAX-WS Bindings Customization**, and then click **Next**. This opens the New JAX-WS Bindings File wizard.
3. Specify the name and location for your bindings file, and click **Next**. This opens the New JAX-WS Bindings File dialog. Select the file from the list of WSDL files available in the project in which the bindings file will be generated.
4. Click **Finish** on the New JAX-WS Bindings File dialog to complete generation of the bindings file in the specified location.

Upon the completion, the new bindings open in the JAX-WS Bindings editor. The contents of the file is a basic bindings file shell with a `wSDLLocation` element pointing to the identified WSDL file.

Once created, you can customize the bindings file using the customizations editor. For more information, see [Section 7.1.1.1, "Customizing a Web Service."](#)

The customizations editor lets you define the contents of a JAX-WS external bindings file (not embedded in the WSDL file), while providing information on the possible XML elements within the file. Using the editor, you can specify a variety of customization options based on the shape of the generated Java code. Note that the editor allows you to define both server-side and client-side customization options.

The customizations editor shows which classes, methods, and parameters will be generated when you apply a particular bindings file's customizations.

You can use the to modify the following types of files:

- A single WSDL file: the editor supports having all WSDL information defined within a single WSDL file. The editor will not load a secondary WSDL file imported through WSDL import or include mechanisms.

- External bindings files: even though you can define JAX-WS bindings elements either within a WSDL file, or in an external file, the customizations editor only supports external bindings files, ignoring the JAX-WS bindings elements within the WSDL file. Note that, although ignored by the editor, those elements will not be ignored by the artifact generation tooling. Also note that you can use the editor to modify only one external bindings file at one time.

You can use the external bindings file to customize the shape of the generated Web service, customize specifics of the generated Web service client (for example, enable asynchronous clients), and add JAX-WS handlers.

- Schema configurations: JAX-WS uses W3C Schema (XSD) elements to define the JAXB types that will be used by the generated Service Endpoint Interface-based Java Web service. You can include the schema information in the WSDL file either in line, or through different import or include mechanisms. The customization editor supports having WSDL type information defined through inline schemas, as well as external schemas imported using the schema import mechanism.

This example shows how inline types define the JAXB type information in an embedded schema element in the types section of the WSDL file.

```
<wsdl:types>
  <xsd:schema elementFormDefault="qualified"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema" targetNamespace="SOME_URL">
    <!-- DEFINE YOUR SCHEMA ELEMENTS HERE -->
  </xsd:schema>
</wsdl:types>
```

This example demonstrates the case of imported schemas, where the customizations editor supports importing schemas directly from the WSDL file using the schema import mechanism. The import element must define a namespace and a schemaLocation attribute. Note that imports in the imported schemas will not be followed.

```
<wsdl:types>
  <xsd:schema>
    <xsd:import namespace="NAMESPACE_OF_SCHEMA" schemaLocation="RELATIVE_PATH_TO_SCHEMA"/>
  </xsd:schema>
</wsdl:types>
```

The customization editor consists of the following parts:

- A tree control on the left - its nodes represent a combination of the WSDL and the Java artifacts that will be generated.
- An edit page on the right - when you select a node in the tree, the editor will display an appropriate edit page where you can edit the properties (supported customization options), as follows:
 - The Global node and the corresponding page let you specify the following settings that apply to the editor in global scope (for example, to all generated Java artifacts):
 - * WSDL URI - the URI to the WSDL file that this bindings file customizes. Note that, even though the JAX-WS 2.1 specification states that the URI can specify a local WSDL or a remote WSDL file, this editor supports only local WSDL files.
 - * Package name and JavaDoc - the package name for the generated Java artifacts. A light-gray value in the name setting means that the default package name will be used. This default value is determined by the WSDL

file's target namespace. The JavaDoc can only be specified if a package name other than the default is specified.

- * Wrapper Style (`enableWrapperStyle` customization option) - the wrapper style setting for Java method generation. Accepting the default value of true will result in wrapper style method generation only if the rules are satisfied, as defined in section 2.3.1.2 of the JAX-WS 2.1 specification, available at <http://jcp.org/aboutJava/communityprocess/mrel/jsr224/index2.html>. This value applies to all endpoint interfaces and methods, however, each endpoint interface or method can override this setting.
 - * Async Mapping (`enableAsyncMapping` customization option) - the async mapping setting for asynchronous method generation on the endpoint interfaces. The default value is true. The generated asynchronous methods can only be used on the client side. This value applies to all endpoint interfaces and methods, however, each endpoint interface or method can override this setting. For more information, see section 2.3.4.2 of the JAX-WS 2.1 specification, available at <http://jcp.org/aboutJava/communityprocess/mrel/jsr224/index2.html>.
 - * MIME content (`enableMIMEContent` customization option) - the MIME content setting for the use of the mime:content information. The default value is true. The client- and server-side settings must match. This value applies to all endpoint interfaces and methods, however, each endpoint interface or method can override this setting. For more information, see section 2.6.3.1 of the JAX-WS 2.1 specification, available at <http://jcp.org/aboutJava/communityprocess/mrel/jsr224/index2.html>.
- The Handler Chains node and the corresponding page, shown in the following figure, display existing, or let you define new handler chains consisting of either logical handlers, SOAP handlers, or a combination of both types. Note that Handler Chains sections on each page (if applicable) provide the same options. You can specify the following:
- * Type - A handler chain can apply to everything (Global), Services, Ports, or Protocol bindings. While this field is not editable in the editor design view, it can be changed in the source view, and the source changes will be reflected in the editor.
 - * Applies To - the specific pattern (Services, Ports), bindings (Protocol), or global (All) to which this handler chain applies. While this field is not editable in the editor design view, you can modify it in the source view, and the source changes will be reflected in the editor.
 - * Class name - Java class that implements `LogicalHandler` interface (for logical handlers), or `SOAPHandler` interface (for SOAP handlers).
 - * Handler name - the name of the Handler that is unique within the module.
 - * Initialization parameters for the Handler with editable names and values
 - * Connection and/or binding setting
 - * SOAP handler settings:
 - SOAP headers - information to identify which SOAP headers will be processed by the SOAP Handler.

SOAP roles - SOAP roles for the SOAP Handler. For more information, see section 10.1.1.1 of the JAX-WS 2.1 specification, available at <http://jcp.org/aboutJava/communityprocess/mrel/jsr224/index2.html>.

- The Endpoint Interface node and the corresponding page, shown in the next figure, represent the endpoint interface that will be generated from a WSDL Port Type after running WSDLC. This node lets you specify:
 - * Interface name and JavaDoc - the name for the generated Java interface. A light-gray value in the name setting means the default interface name is being used. This default value is determined by a WSDL Port Type. You can only specify the JavaDoc if the specified interface name is other than the default.
 - * Wrapper Style (enableWrapperStyle customization option) - the wrapper style setting for Java method generation on a specific endpoint interface. The default value is determined by the Wrapper Style setting on the Global node. This value applies to a particular endpoint interface and its methods, however, each method can override this setting.
 - * Async Mapping (enableAsyncMapping customization option) - the async mapping setting for asynchronous method generation on a specific endpoint interfaces. The default value is determined by the Async Mapping setting on the Global node. This value applies to a particular endpoint interface and its methods, however, each method can override this setting.
 - * Services - represents the implementing class that will be generated from a WSDL Port after running WSDLC:
 - Service class name and JavaDoc - the name for the generated Java class. A light-gray value in the name setting means the default class name is being used. This default value is determined by a WSDL Port Type. You can only specify the JavaDoc if the specified interface name is other than the default.
 - MIME content (enableMIMEContent customization option) - the MIME content setting for the use of the mime:content information on the specific Service. The default value is determined by the MIME Content setting on the Global node. The value in the editor screen applies to the particular Service only.
- The Method node and the corresponding page, shown in the next figure, represent the method that will be generated from a WSDL operation after running WSDLC. This node lets you specify:
 - * Method name and JavaDoc - the name for the generated Java method. A light-gray value in the name setting means the default method name is being used. This default value is determined by a WSDL operation. You can only specify the JavaDoc if the specified method name is other than the default.
 - * Wrapper Style (enableWrapperStyle customization option) - the wrapper style setting for Java method generation on a specific method. The default value is determined by the Wrapper Style setting on a parent Endpoint Interface node. This value applies to a particular method only.
 - * Async Mapping (enableAsyncMapping customization option) - the async mapping setting for asynchronous method generation on a specific method. The default value is determined by the Async Mapping setting on

a parent Endpoint Interface node. This value applies to a particular method only.

- * Method parameters - represents the method's parameters and the customization that will apply on the parameter name. Only the name is editable.
- The Faults node and the corresponding page will only appear if there are Faults on the WSDL operation. This node serves as a grouping node to show a list of all the defined Faults. This node lets you specify:
 - Fault Class name and JavaDoc - the name for the generated Java class. A light-gray value in the name setting means the default class name is being used. This default value is determined by a WSDL fault. You can only specify the JavaDoc if the specified class name is other than the default.

Notice that the editor pages open with default values already specified.

7.1.2 Generating a Web Service From Java

OEPE lets you generate Web services from Java using the following starting points:

- From a Java class
- From nothing

7.1.2.1 Creating a Web Service from a Java Class

You can develop a JAX-WS-enabled Web service as a Java class, with methods becoming Web service operations, and method parameters and return types being Java beans. To indicate what methods should be exposed and to set other properties for the service, you use annotations.

To create a Web service from a Java class:

1. Right-click the class file, and select **WebLogic Web Services > Generate Web Service**. This opens the Generate Web Service from Existing Java Class wizard.
2. On the New JAX-WS Web Service page, enter the **Web Service Name** (corresponding to the `@WebService serviceName` attribute), the **Port Name** (corresponding to the `@WebService portName` attribute), and select **Add SEI** to optionally generate the endpoint interface.
3. On the Message Format page, select the **SOAP Binding** (default: 1.1), the **SOAP Message Format** (default: Document/Literal/Wrapped), and select **Enable MTOM** to optionally implement MTOM binary encoding.
4. On the Methods page, select methods for the web service to expose.
5. On the Selection WebLogic Web Service Policies page, select either **No Policies**, **OWSM Web Service Policies**, or **WebLogic Service Policies**.

Note: You cannot mix OWSM and WebLogic Policies.

The Description section provides information about the currently selected policy. As you select available policies, they are validated against each other for compatibility. If incompatible policies are selected, a validation error is displayed. If you click on the error marker, text describing the validation error is displayed. Then click **Finish**.

See [Section 7.1.7, "Understanding Policy Stores"](#) for more information on configuring policy stores.

See [Section 7.6, "Using WebLogic Web Services Annotations View"](#) for information on adding and removing OWSM Web Service Policies and WebLogic Service Policies.

Having followed the preceding procedure, you created a Web service from a Java class. The resulting generated code has imports added, and the class annotated corresponding to wizard choices. You can run it by right-clicking the new class, and then selecting **Run As > Run on Server** from the drop-down menu.

Web services without security policies can be tested using the Test Client. For more information, see [Section 7.1.8, "Testing Web Services"](#)

7.1.2.2 Creating a Web Service From Scratch Using Java

Using OEPE, you can create a JAX-WS-enabled Web service for your Java project without taking any prior steps.

To create a Web service from scratch:

1. Create a new Web service project.
2. Right-click the project name in the Project Explorer and select **New > Other** from the drop-down menu. This will open the New wizard. Select **Oracle > WebLogic > Web Services > WebLogic Web Service**, and then click **Next** to open the New Web Service wizard.
3. On the Web Service page, provide a **Package** name and enter a **Name** for your new Web service. Optionally, click **Enable MTOM** to implement binary encoding. Optionally, click **Generate Service Endpoint Interface** to generate an endpoint interface. Click **Next**.
4. On the Selection WebLogic Web Service Policies page, select either **No Policies**, **OWSM Web Service Policies**, or **WebLogic Service Policies**.

Note: You cannot mix OWSM and WebLogic Policies.

The Description section provides information about the currently selected policy. As you select available policies, they are validated against each other for compatibility. If incompatible policies are selected, a validation error is displayed. If you click on the error marker, text describing the validation error is displayed. Then click Finish.

See [Understanding Policy Stores](#) for more information on configuring policy stores.

7.1.3 Generating a WSDL File

Using OEPE, you can generate a WSDL file from a Java class for your project.

To generate a WSDL:

1. Create new or use an existing Web service project.
2. Right-click a Java class in your Web service project in the Project Explorer, and select **WebLogic Web Services > Generate WSDL** from the drop-down menu.

This generates a WSDL file in the same package as the Java class that you used as a starting point.

7.1.4 Contents of a WSDL File

Files with the WSDL extension contain Web service interfaces expressed in the Web Service Description Language (WSDL). WSDL is a standard XML document type specified by the World Wide Web Consortium (W3C). For more information, see <http://www.w3.org/TR/wsdl>.

WSDL files communicate interface information between Web service producers and consumers. A WSDL description allows a client to utilize a Web service's capabilities without knowledge of the implementation details of the Web service.

A WSDL file contains the following information necessary for a client to invoke the methods of a Web service:

- The data types used as method parameters or return values.
- The individual methods names and signatures (WSDL refers to methods as operations).
- The protocols and message formats allowed for each method.
- The URLs used to access the Web service.

7.1.5 Imported WSDL Files

When you want to use an external Web service from within Eclipse, you should first obtain the WSDL file for the service you want to use, as follows:

- For public Web services, the WSDL file will typically be available on the Web site of the organization that publishes the Web service.
- For private Web services, contact the organization that supports the Web service to obtain the WSDL file.

WSDL files can also be found through both public and private UDDI registries. For more information about UDDI, see <http://uddi.xml.org/>.

Once you have the WSDL file, you may use Eclipse to create a Web service.

Some Web service tools produce WSDL files that do not contain an XML declaration. The XML declaration is just the first line of an XML file of the following form:

```
<?xml version="1.0" encoding="utf-8" ?>
```

If you receive a WSDL file that does not contain an XML declaration, you must add a declaration to the file using a text editor before you can use the WSDL file in Eclipse.

Note: The encoding attribute is not required. If an encoding attribute is not present, the default encoding is `utf-8`.

7.1.6 Creating a New WSDL File

To create a WSDL file to use in your project:

1. Right-click the project in the Project Explorer and select **New > Other** from the drop-down menu. This opens the New dialog.
2. In the New dialog, select **Web Services > WSDL**, and then click **Next**. The New WSDL File dialog opens.
3. On the New WSDL File dialog, provide a name for the WSDL file, click **Next**, and then click **Finish**. This creates a WSDL file for your project.

Note: For a JAX-WS Web service project, you enable the Standard Annotated Web Services project facet option by right-clicking the project in the Project Explorer, selecting **Properties** from the drop-down menu, and then selecting **Project Facets > Oracle WebLogic Web Services**.

7.1.7 Understanding Policy Stores

Use to specify the policy store location for the WS Policy Store, which is where the Oracle Web Services Manager (Oracle WSM) policies are defined.

Policies, which are used to provide management and security functionality, are created and managed centrally in an organization by the web service policy manager or security architect, who provisions a set of policies using Oracle WSM. Policies can be updated without needing to modify the web services that use the policies.

OEPE comes with a default file-based policy store, which is part of Oracle WebLogic Server, which means that you can use policies out of the box. You apply policies in the Create Web Services wizards.

To configure the policy store in OEPE:

1. From the main menu, select Window > Preferences to open the Preferences dialog.
2. In the Preferences dialog, select WebLogic > WS Policy Store.
3. On the WS Policy Store page:
 - Click a Runtime to configure its corresponding policy store.
 - File Store: Click to use the default file-based policy store.
 - Override: Click to enter or browse to the fully-qualified path to the location of the policy store to be used in place of the default policy store, in the Override Location field.
 - Remote WebLogic: Click to enter the remote location of the WebLogic Server instance that contains the policy store.

For more information on policies in other contexts, see *Securing WebLogic Web Services for Oracle WebLogic Services*.

7.1.8 Testing Web Services

As you develop a Web service, you can test it directly by using the test client. The test client provides a user interface through which you can test Web service operations with parameter values you choose.

Using the test client, you can do the following:

- **Test a Web service from the project tree:** When you test Web services with Eclipse, consider the following steps that launch the test client with a visual interface for invoking the Web service's operations:
 1. Expand the project tree to display the Web service source file.
 2. Right-click the source file, then click **Run > Run on Server**.
 3. When the test client is displayed, choose the operation you want to test by clicking the button labeled with the operation's name. If the operation has parameters, the test client provides boxes for you to enter the values to test

with. If an operation includes complex types as parameters, the test client will display an XML template with placeholders for your test values.

4. Examine the result of the test by looking at generated messages. When you execute an operation, the test client refreshes to display information about the message exchanged by the operation. The user interface provides a summary of message values, as well as the message XML itself. When an exception occurs, a fault message is displayed.
 5. Use the Message Log list to view the results of multiple tests.
 6. Click **Show Operations** to begin another test.
- **View the WSDL file for the Web service you are currently testing:** click the WSDL URL provided at the top of the test client window.
 - **Choose another Web service to test:** You can test another Web service without closing the test client by clicking the Choose Another WSDL link at the top of the test client window. The test client will display a page with a box where you enter the WSDL URL, then click Test to display the test form for that Web service.

Alternatively, you can launch the test client without using Eclipse IDE by launching the client through a Web browser, as follows:

1. With the Oracle WebLogic Server running, open a browser window and navigate to `http://localhost:7001/wls_utc` to start the test client.
2. In the Enter WSDL URL box, enter the URL for the WSDL of the Web service you want to test, and then click Test.

7.2 Creating Web Services Projects

Use web service projects to develop web services that conform to standards, such as SOAP for message exchange, XML for messages to or from the service, and a WSDL that specifies the web service's public interface.

Each Web service project produces a JEE module, each of which is included in the complete Eclipse application's EAR file when you build your application for deployment.

The contents of Web service projects are accessed through the test client which allows you to access each operation of the Web service by clicking on a button. For more information, see [Section 7.1.8, "Testing Web Services."](#)

7.2.1 Creating a new Web Service Project

Using OEPE, you can create a Web service project.

To create a Web service project:

1. Right-click the Project Explorer and select **New > Project** from the drop-down menu. This opens the New Project dialog.
2. Select **WebLogic > Web Services > Web Service Project** from the list, and then click Next.
3. On the New Web Service Project dialog, provide a name and location for your project, select and configure your target runtime, specify your EAR and working sets settings, and then click **Finish**.

7.2.2 Creating a Web Service Project From an Existing Dynamic Web Project

Using Eclipse IDE and Oracle WebLogic Server, you can create a Web service project from an existing dynamic Web project by adding a Web Services facet.

Dynamic Web projects are used to create Web applications.

User interface components are constructed from Java Server Pages (JSPs), which are Web pages that can interact with server resources to produce dynamic content.

Each dynamic Web project ultimately produces a J2EE module. Each J2EE module is included in the complete application's EAR file when the application is built for deployment.

The contents of Web projects are accessed through URLs.

A dynamic Web project has two core facets: the Dynamic Web Module (the enable facet) and the WebLogic Web App Extensions facets. This project may contain a number of optional facets, one of which is the Web Service facet.

To create a Web service project from an existing dynamic Web project:

1. Open your dynamic Web project in Project Explorer.
2. Right-click on the project name and select **Properties** from the drop-down menu. This will open the Properties dialog.
3. Select **Project Facets** from the tree-control on the dialog's left panel.
4. Select **WebLogic Web Services** from the Project Facet list on the dialog's right panel.
5. Click **Apply**.

7.3 Generating Client Code for Web Services

You can generate client code to use with web services.

7.3.1 Generating Client Code From a WSDL File

Using OEPE, you can generate the client code for the following types of projects:

- Web service project
- Dynamic Web project
- EJB project

To generate the client code for your Web service:

1. Create new, or obtain an existing WSDL file for your target Web service project.

For more information, see the following:

- Creating a new WSDL File
 - Generating a Web Service from a Java Class for JAX-WS
2. Create another Web service project, which will be your client project.
 3. Copy the WSDL file (from step 1) located in the WebContent folder of your target Web service into the WebContent folder of your client project (from step 2).
 4. Generate the client code as follows:

- In the Project Explorer, right-click your client project's WSDL file, and then select **WebLogic Web Services > Generate Web Service Client** from the drop-down menu. This will open the New Web Service Client dialog.
- On the New Web Service Client dialog, specify the service name, the output location of the generated JAR file, as well as Ant script generation options, and then click **Next**.
- On the next New Web Service Client > Customization Options screen, you may choose to specify additional options, and then click **Finish**.

Upon completion of the preceding procedure, Eclipse does the following:

- Creates the client JAR file in your client Web service project's `WebContent/WEB-INF` directory.
- Modifies properties of your target Web service project, as necessary (sets the project facet, classpath, and so on).

If you deploy and run both your target and client Web service projects, your generated client code will successfully call your target Web service.

Note: Nondeployment-based applications, such as Java applications, may require additional manual configuration.

You can reference the client from the target Web service using the template-based code snippets.

7.3.2 Generating Client Code from a Java Class

You can generate client code from Java classes.

To generate client code from a Java class:

1. Choose **File > New > Other**. In the New Dialog expand **Oracle > WebLogic > Web Services** and choose **Web Service Client Invocation**.
2. On the Web Service Proxy Client Generation page, populate the following fields:
 - **Project:** Choose the project that will contain the proxy class as well as the class you want to generate the client invocation code from.
 - **Client Proxy Class:** Browse the current project for an existing WebService client, or choose to create a new webservice client using a local or remote WSDL.
 - **Invocation Class:** Choose a class file that you want to generate the client invocation into, or browse for an existing class.
 - **Invocation Method:** Choose the corresponding method that will contain the client invocation. The resulting code block will be generated at the end of the method block. You can also browse for existing methods.
3. On the Selection WebLogic Web Service Policies page, select either **No Policies**, **OWSM Web Service Policies**, or **WebLogic Service Policies**.

Note: You cannot mix OWSM and WebLogic Policies.

Within the OWSM policy panel, you can choose compatible client policies based on a resolved WSDL location. The WSDL location is based on the client service

selection specified in the previous page (Web Service Proxy Client Generation). If the WSDL is packaged with the service, OEPE evaluates the WSDL for policy declarations and a compatible client list is displayed. Otherwise, you are presented with the entire client store of policies.

As you select available policies, they are validated against each other for compatibility. If incompatible policies are selected, a validation error is displayed. If you click on the error marker, text describing the validation error is displayed. You will be required to resolve the validation errors prior to selecting **Finish**.

7.3.3 Alternative Ways to Generate the Client Code

There are alternative entry points to the client code generation wizard. Instead of right-clicking your client project's WSDL file, and then selecting **WebLogic Web Services > Generate Web Service Client** from the drop-down menu, you can also right-click the Project Explorer area, select **New > Other > WebLogic Web Services > Web Service Client**, and then click **Next**. This opens the **ClientGen Wizard > WSDL File** dialog.

Using this dialog, select the WSDL file from which you want to generate the Web service client indicating if this file is local or remote (in this case, enter a remote URL), and then proceed with your other settings.

7.3.4 Deploying Java Web Service Applications to Oracle WebLogic Server

Typically, Eclipse builds your files automatically and you can deploy your application at any time. Note that in order to deploy, you must have a successful build completed.

You can start Oracle Weblogic Server in either development or production mode: in development mode, the server behaves in ways that make it easier to iteratively develop and test applications (for example, the server automatically deploys the current application in an exploded format and relaxes certain security restrictions on deployment); in production mode, WebLogic Web Service test client is not deployed. For more information, see *Testing Web Services*.

Application deployment consists of the following three steps:

1. Compilation of the Web service.
2. Publishing the files to the server.
3. Running the application.

Usually these steps are done seamlessly, by explosion of the WAR file.

When you deploy, all open projects associated with the server are deployed. To undeploy a project or prevent it from being deployed, remove it from the server. Alternatively, you can close projects by selecting **Project > Close Projects** from the menu. You can also specify working sets (see Eclipse help system in the *Workbench User Guide*) to control how many artifacts to build and deploy.

However, even though an entire application or group of projects was deployed, only the file or folder that you clicked on to trigger the deployment will display its results. When you deploy your Web service, the test client page for that Web service will run in a new tab in the editor area - you can use it to specify the parameters to an operation and make a request to that operation. The response from the operation is displayed in the same tab. If the page flow or Web service relies on other Web services to run correctly, it will still work because all components are deployed.

To deploy your Web service application, do one of:

- Right-click your project in the Project Explorer and select **Run As > Run on Server** from the drop-down menu to start the server in normal mode (if it has not been started). This will deploy the application to its associated server; if the project is not associated with a server, make a server selection using the displayed dialog, and then run the application.
- Right-click your project in the Project Explorer and select **Debug As > Debug on Server** from the drop-down menu to start the server in debug mode (if it is not already started), and run the application.

If you wish to simply publish your files to the server, you can use the Servers view by selecting **Window > Show View > Servers** from the main menu.

Note: This view also lets you undeploy applications from the server.

After you deploy your application, the Servers view is displayed automatically.

7.4 Generating JAXB Types

Using OEPE, you can generate JAXB classes from an existing XML schema for your Web service project.

To generate JAXB classes:

1. In the Project Explorer, right-click the schema file (XSD file) in your Web service project, and then select **WebLogic Web Services > Generate JAXB Types** from the drop-down menu. This will open the New JAXB Types > Generated Artifacts dialog.
2. On the New JAXB Types > Generated Artifacts dialog, specify the output location of the generated JAR file, as well as the Ant script generation options, and then click **Next**.

Note: You can also save the generated Ant script for future modifications to the generation of the JAXB artifacts. To do so, select **Keep generated Ant script** to save the file.

3. On the next New JAXB Types > Customization Options screen, you may choose to specify additional options, and then click **Finish**.

Upon completion of the preceding procedure, Eclipse creates the JAXB type JAR file in your Web service project's `WebContent/WEB-INF/lib` directory.

For information on using generated JAXB types with your Web service, see *Web Services Development: Starting Points for JAX-WS*.

7.5 Using Client Proxy Templates

OEPE provides a set of ready-to-use common code snippets that you can access through templates and insert in your JAX-WS-enabled Web services Java code. This is useful when you instantiate a Web service client.

To insert a code snippet:

1. Within your Web service project, open a Java file to which you want to add the code.

2. Place the cursor at a particular location, and then press **Ctrl+Space**. This opens a descriptive list of titles of the code snippets available for your code. To display a detailed description and the snippet itself, select an item from the list.
3. Double-click the snippet title to insert it in your code.

Using the following automatically-generated code snippets, you can reference new or modify an existing Web service client:

- Default WSDL location client Web service - use this option to create a basic Web service client reference.

The generated code will be of the following format:

```
SomeService service = new SomeService();
SomePortType port = service.getSomePort();
port.someOperation();
```

- Specify WSDL location client Web service - use this option to create a basic Web service client reference, with the local declaration allowing for the URL specification.

The generated code will be of the following format:

```
// variable declaration - implementation injected by container

@WebServiceRef(wsdlLocation="http://localhost:7001/SomeProject/SomeService?wsdl")
SomeService service;

// Web service call in the method block
SomePortType port = service.getSomePort();
port.someOperation();
```

- Web service reference client service - use this option to create a basic Web service client reference field declaration, where you will be required to specify the URL.

The generated code will be of the following format:

```
SomePortType port = service.getSomePort();

Binding binding = port.getBinding();

// can create new list or use existing one
List handlerList = binding.getHandlerChain();

handlerList.add(new HandlerChainImplementation());
binding.setHandlerChain(handlerList);
port.someOperation();
```

- Attach programmatic client-side handler chain - use this option to attach a handler chain programmatically for a locally declared client service.

The generated code will be of the following format:

```
SomePortType port = service.getSomePort();

Binding binding = port.getBinding();

// can create new list or use existing one
List handlerList = binding.getHandlerChain();

handlerList.add(new HandlerChainImplementation());
binding.setHandlerChain(handlerList);
```

```
port.someOperation();
```

- Web service reference client service with a handler chain - use this option to create a basic Web service client reference field declaration, where you will be required to specify the URL and a handler chain file location.

The generated code will be of the following format:

```
// variable declaration - implementation injected by container
@javax.jws.HandlerChain(file="myhandler.xml")

@WebServiceRef(wsdlLocation="http://localhost:7001/SomeProject/SomeService?wsdl")
SomeService service;

// Web service call in the method block
SomePortType port = service.getSomePort();
port.someOperation();
```

Note: With the exception of the template that you use to programmatically attach a handler chain for a locally declared client service, all other templates require a WebLogic Web service-generated client JAR with attached source located in the `WebContent/WEB-INF/lib` directory.

7.6 Using WebLogic Web Services Annotations View

You use the WebLogic Web Services Annotation view to add new and edit existing annotations of a Java Web service within projects configured with WebLogic Web Services or WebLogic Web Service Clients facet. This view allows you to add annotations and their attributes without knowing the detailed information about which annotations and attributes are supported.

7.6.1 Activating the WebLogic Web Services Annotations View

To activate the WebLogic Web Services Annotation view:

1. In the Project Explorer, open one of the source Java files from your project that contains a Web service. Use the Java code editor to open the file.
2. Select `Window > Show View > Other` from the top menu to open the Show View dialog. Select `Web Services > WebLogic Web Service Annotations` from the list, and then click OK to open the view.

The figure below shows the activated WebLogic Web Service Annotations view that makes the annotation selected in the Java source available for editing.

7.6.2 Using the WebLogic Web Services Annotations View

You use the WebLogic Web Services Annotations view as you would any other editor.

When using the WebLogic Web Services Annotations view, consider the following features:

- When you select a valid Java element in the Java editor, a set of tabs is displayed, with each tab representing a supported annotation.

- When you select an annotation in the Java editor, the corresponding tab is activated.
- Each tab provides a property editor pertaining to the attributes for that particular annotation:
 - Each tab displays a link action to either Add or Remove the annotation from the Java source.
 - Annotations are automatically added when you start to edit an attribute on the property editor.
 - Annotations are automatically removed if all attributes are cleared from the property editor.
- When you edit an attribute, the changes are immediately reflected in the Java source editor.
- Edits in the Java source editor are immediately reflected in the WebLogic Web Services Annotations view's property editor.
- You can clear the value or return to a default value by using the light bulb icon next to a field.
- Default values are displayed in grey.

7.6.3 Supported Annotations

You can use the WebLogic Web Services Annotations view to edit the following annotations:

- Class-level annotations:
 - WebService
 - SOAPBinding
 - HandlerChain
 - WebFault
 - ServiceMode
 - WebServiceClient
 - WebServiceProvider
 - BindingType
 - WebServiceRefs
 - SecurityPolicies
 - SecurityPolicy
 - Policies
 - Policy
- Field-level annotations:
 - WebServiceRef
 - HandlerChain
- Method-level annotations:
 - SOAPBinding

- WebMethod
- Oneway
- WebResult
- WebEndpoint
- RequestWrapper
- ResponseWrapper
- Parameter-level annotations:
 - WebParam

Note: You can use some annotations, such as `WebService`, `WebMethod`, and `Oneway`, as marker annotations (no attributes are required when the annotation is specified).

7.7 Validating Web Services Projects

OEPE provides a validation utility for your WebLogic Web services projects that you develop in Eclipse IDE. Subject to validation are various artifacts and associations within the project, including compliance with specifications (for example, JSR-224), verification of referenced resources (for example, local and remote), and so on. The validation occurs by the means of a standard Eclipse validator that indicates errors in the Problems view, in the Project Explorer, and as markers in the source view, as Figure 1 and Figure 2 show.

7.7.1 Validated Resources

When working with Java Web services in wls.web faceted projects, you can expect the following types of errors to be validated:

- References:
 - annotations, with the following validation notes:
 - * `WebService` - 1) Evaluate if name is specified as an attribute, in case of explicit service endpoint interface; 2) Evaluate if `wSDLLocation` declared differently on service endpoint interface and service implementation; 3) Evaluate if the referenced endpoint interface has the `WebService` annotation; 4) Evaluate if the referenced endpoint interface is an interface.
 - * `WebMethod` - 1) Methods are not overridden; 2) Methods are public; 3) Method return type is not `XmlBeans`; 4) Method parameters are not `XmlBean`; 5) Non-`WebMethod` with `WebService` annotations;
 - * `SoapBinding` - RPC encoded is not supported.
 - * `Exclude` - Evaluate methods with this annotation for exclusion from `WebMethod` validation.
 - * `BindingType` - `HTTPBinding` specified.
 - * `WebServiceProvider` - Attribute `wSDLLocation` is required for Provider-based Web service when `HTTPBinding` type is not specified.
- Runtime artifacts (context path):
 - Oracle WebLogic Server-specific errors.

- Generated artifacts in synchronization.

7.7.2 Configuring Project Validation

To enable or disable validation:

1. Right-click your Web services project in the Project Explorer and select **Properties** from the drop-down menu. This opens the Properties dialog, as Figure 3 shows.
2. In the Properties dialog, expand the **Validation** node. The editor pane on the right displays all available validators whose settings you can modify (notice **Web Services for WebLogic**). From this pane, you can do the following:
 - Enable settings specific to your project.
 - Enable each particular setting.
 - Enable or disable all validators from the list.
 - Suspend validation.
 - Configure your workspace settings: this option becomes available if you select **Suspend all validators**, and then click **Configure Workspace Settings**. This opens the expanded Preferences > Validation dialog that Figure 4 shows.

Note: The default validation filters enable validation of Web services within a WebLogic Web services project (for example, those with the `oracle.weblogic.webservices` and `oracle.weblogic.webservice.clients` facets).

7.8 Generating Web Services for Spring Service Beans

You can generate a WebLogic Web Service from an existing Spring service bean for your JAX-WS Web service project.

To generate the Web service:

1. Add the WebLogic Web Services facet to an Eclipse project configured with the Spring 2.5 facet and containing Spring service beans. For more information about configuring projects for Spring, see *Generating Spring Artifacts*.
2. In the Project Explorer, right-click your project, select **New > Other** from the drop-down menu, and then select **Oracle > WebLogic > WebLogic Web Service for Spring Beans** on the New dialog. This opens the Select Spring Service Bean dialog.
3. On the Select Spring Service Bean dialog, specify the Spring bean configuration. The list will be populated with Spring service beans defined in the configuration file. Select a service bean and service methods from the service interface that you want to expose as WebLogic Web Service, and then click **Next**.

Note: While one service bean can implement multiple interfaces, you can only select methods from one interface at a time. To expose methods in other interfaces, you need to generate a separate Web service. Also note that Spring ORM service beans implemented in Spring 1.*n*-style based on the `JpaTemplate` and `TransactionProxyFactoryBean` are not supported, therefore it is recommended that you upgrade your service bean to Spring 2.0-style using annotation-based transaction management.

4. On the next page, Web Service Class Name dialog, shown in Figure 4, you may choose to specify the Web service class name, package name and output directory, and then click **Finish**.

Upon completion of the preceding procedure, Eclipse creates the WebLogic Web Service class which is automatically wired up with Spring bean and delegate service calls to the Spring bean implementation.

7.9 Configuring HTTPS Client Credentials

Use the HTTPS Client Credentials page to specify the keystores Eclipse should use when handling HTTPS traffic.

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. Use the HTTP Client Credentials tab to enter details of the certificate keystore for the client. The client keystore identity is used for configuring HTTPS.

For more information about keystores and keystore providers, see *Understanding Security for Oracle WebLogic Server*.

To configure credentials:

1. From the main menu, choose **Window > Preferences**.
2. In the Preferences dialog, select **WebLogic > Credentials**.
3. On the Credentials page, select a WebLogic Server runtime to configure its keystore and credential information or add a new runtime.
4. With a runtime selected, configure its HTTPS Client Credential information in the **HTTPS Client Credentials** section:
 - **Client Trusted Certificate Keystore:** Enter, or browse to, the location of the client certificate keystore.
 - **Client Trusted Keystore Password:** Enter the password for the client keystore. The default is DemoTrustKeyStorePassPhrase.
 - **Edit Keystore:** Click to add or remove entries in the keystore.
 - **Client Keystore:** Enter or browse to the location of the client keystore.
 - **Client Keystore Password:** Enter the password for the client keystore.
 - **Client Private Key Password:** Enter the client private key password.
 - **Append default CA certificates:** Select to add the CA certificates.
 - **Restore Defaults:** Click to restore default settings.

REST Web Services Support

This chapter describes how to create and run web services in OEPE which conform to the Representational State Transfer (REST) architectural style using Java API for RESTful Web Services (JAX-RS).

This document contains the following sections:

- [Getting Started with REST Web Services](#)
- [Creating Projects Configured for REST](#)
- [Creating a REST Web Service](#)
- [Mapping Incoming Requests to Java Methods](#)
- [Customizing Media Types for the Request and Response Messages](#)
- [Validation and Quick Fix](#)
- [Content Assist](#)
- [Run-AS JAX-RS Support](#)
- [Generate a Java REST Client from a WADL](#)

8.1 Getting Started with REST Web Services

OEPE provides tools to create Representational State Transfer (REST or RESTful) projects, configure Java projects for REST and build REST web services for your projects. 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.

There are a couple steps to enabling REST web services in your OEPE applications:

1. Create a web project:
 - [How to Create a Dynamic Web Project that is Configured for REST](#)
 - [How to Configure a Java Project for REST](#)
2. [Creating a REST Web Service](#)

For complete details about developing REST web services and clients using JAX-RS, see the following Oracle documentation:

- *Developing and Securing RESTful Web Services for Oracle WebLogic Server*

- *Jersey: RESTful Web services made easy*, at <https://wikis.oracle.com/display/Jersey/Main>

8.2 Creating Projects Configured for REST

To use REST in OEPE, first create a project that is configured to support REST (JAX-RS) services. You can do this in a couple steps using the web project wizards in the OEPE interface.

There are two types of projects that you can configure to use REST:

- [How to Create a Dynamic Web Project that is Configured for REST](#)
- [How to Configure a Java Project for REST](#)

8.2.1 How to Create a Dynamic Web Project that is Configured for REST

When you create a dynamic web project that is configured for REST, the system adds a `web.xml` file and a `weblogic.xml` file that contain the REST servlet and servlet mapping detail that your project uses at runtime, as shown in [Example 8–1](#) and [Example 8–2](#).

Example 8–1 *web.xml configured for REST*

```
<display-name>REST Project</display-name>
<servlet>
<description>JAX-RS Tools Generated - Do not modify</description>
<servlet-name>JAX-RS Servlet</servlet-name>
<servlet-class>com.sun.jersey.spi.container.servlet.ServletContainer</servlet-class>
<load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
<servlet-name>JAX-RS Servlet</servlet-name>
<url-pattern>/jaxrs/*</url-pattern>
</servlet-mapping>
</web-app>
```

Example 8–2 *weblogic.xml configured for REST*

```
<wls:weblogic-version>12.1.3</wls:weblogic-version>
<wls:context-root>REST_Project</wls:context-root>
```

To create a dynamic web project configured for REST

1. Create a WebLogic runtime. Both Weblogic 11g and 12c versions are supported, as well as Glassfish. These steps are for Weblogic 12c. For Weblogic 11g, the wizard interface is slightly different, and adds a `weblogic.xml` file with a `jax` library reference `<wls:library-name> jax-rs</wls:library-name>`.
2. In the OEPE application, choose **File > New > Web > Dynamic Web Project**.
3. Click **Next**. The New Dynamic Web Project dialog appears, as shown in [Figure 8–1](#). Add the project detail, and choose the target runtime. The target runtime is shown in the Configuration field.
4. Next to the Configuration field, click **Modify** to choose what capabilities to enable for your project. The Project Facets dialog appears, as shown in [Figure 8–2](#).
5. Select **JAX-RS (REST Web Services)**. Click **OK**.

- Click **Next**. The JAX-RX Capabilities dialog appears with the default servlet and container details. If you accept the defaults click **Finish**.

Note: When you change the class or project class path in the run configuration wizard, be sure to manually stop the JAXRS application and restart it to reflect your changes. Unless it is stopped and started again, it will not reflect the changes.

Figure 8–1 New Dynamic Web Project

New Dynamic Web Project

Dynamic Web Project
Create a standalone Dynamic Web project or add it to a new or existing Enterprise Application.

Project name: REST Project

Project location
 Use default location
Location: C:\Users\peander\REST Project Browse...

Target runtime
Oracle WebLogic Server 12c (12.1.2) New Runtime...

Dynamic web module version
3.0

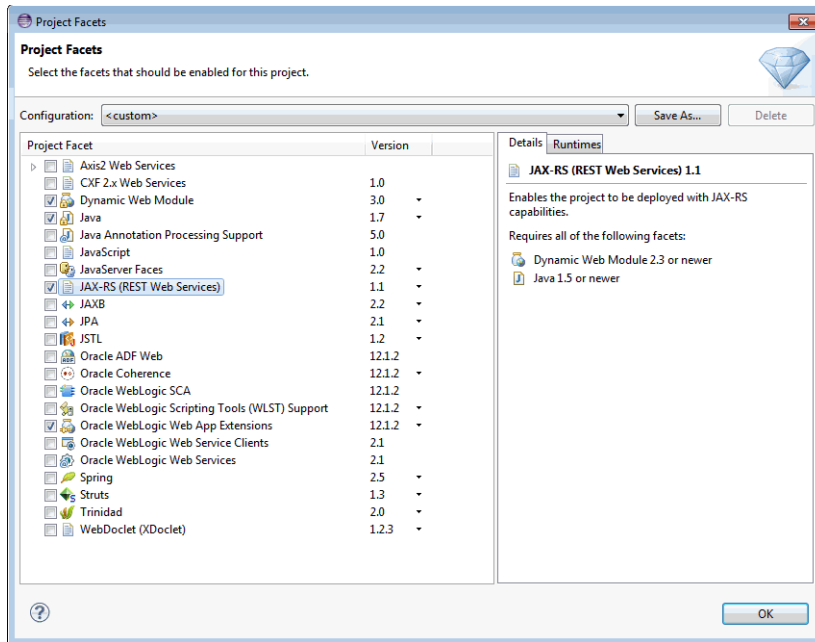
Configuration
Default Configuration for Oracle WebLogic Server 12c (12.1.2) Modify...
A good starting point for working with Oracle WebLogic Server 12c (12.1.2) runtime.
Additional facets can later be installed to add new functionality to the project.

EAR membership
 Add project to an EAR
EAR project name: EAR New Project...

Working sets
 Add project to working sets
Working sets: Select...

? < Back Next > Finish Cancel

Figure 8–2 REST Project Facets



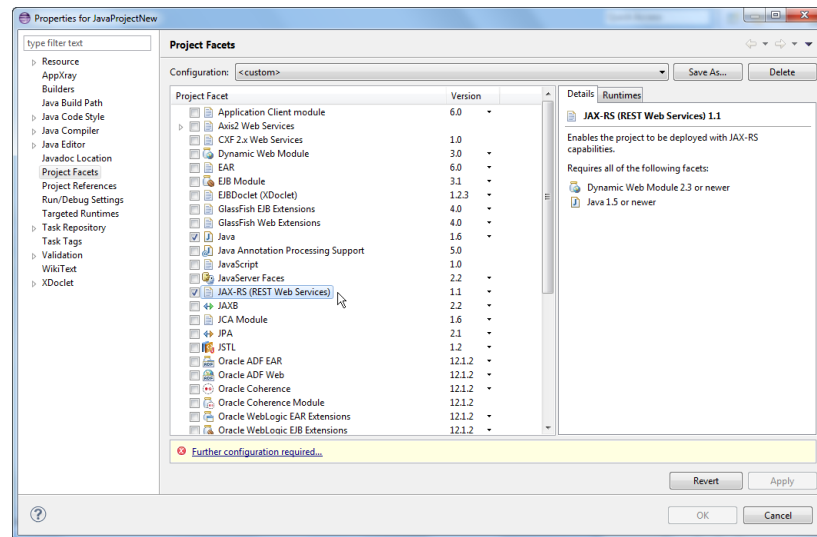
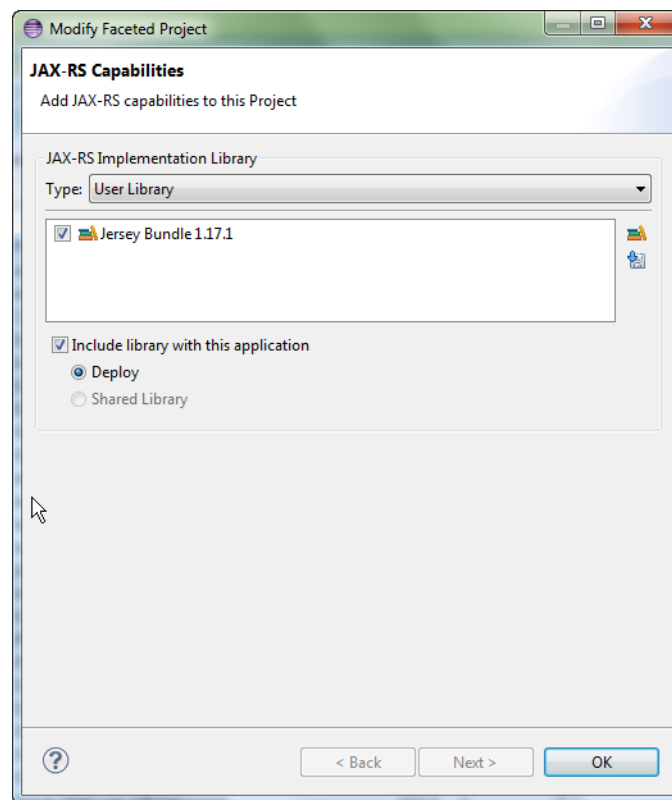
8.2.2 How to Configure a Java Project for REST

To use REST with your Java web project you need to add the JAX-RS library bundle. To do this, convert your Java project to a JAX-RS faceted project.

Note: JAX-RS tooling in a Java Project does not currently work with a Glassfish Runtime Specific Library Provider. Currently, Glassfish requires a Dynamic Web Project for deployment and to access the JAX-RS Implementation library provider.

To convert a Java project for REST

1. Select your Java project. Right-click and choose **Configure > Convert to Faceted Form**.
2. In the Facet Page select **JAX-RS**, and choose the appropriate runtime version, as shown in [Figure 8–3](#).
3. Notice the "Require Further Configuration" link highlighted in the lower part of the dialog. Click Require Further Configuration to continue to the JAX-RS Facet Install page, as shown in [Figure 8–4](#).
4. In the JAXRS Facet Install page, select **User Library**.
5. Select the download button, or select from the previously configured user libraries available for your JAX-RS Runtime, as displayed in the libraries window. The Jersey Download is now available.
6. Click **OK**.

Figure 8–3 Convert a Java Project for REST - Project Facets**Figure 8–4 Convert a Java Project for REST - JAX-RS Capabilities**

8.3 Creating a REST Web Service

Once you have configured your project for REST, you are ready to create your REST web service.

You can create a new REST web service class or generate a REST web service from an existing Java class using the Create RESTful Service wizard. The wizard creates the

deployment files for you. After you create your web service, the final step is to deploy it.

When you create a RESTful web service, the JAX-RS Jersey library is automatically added to your REST project.

You can create are two types of REST web services in OEPE:

- [How to Create a Patterned REST Web Service.](#)
- [How to Create a POJO REST Web Service.](#)

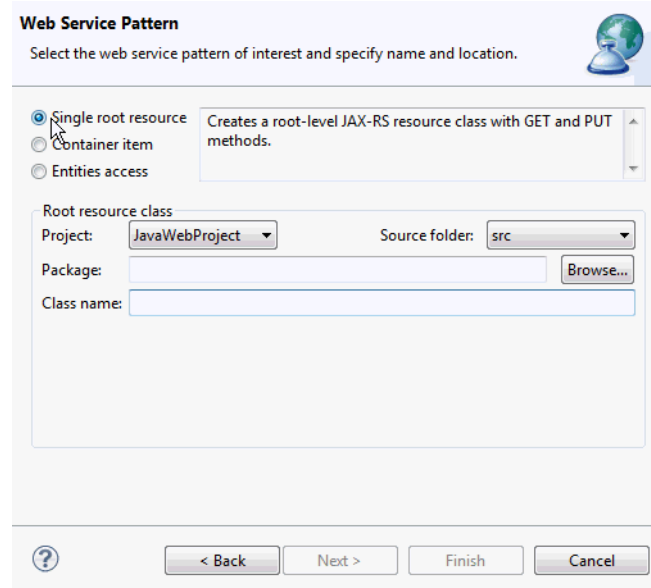
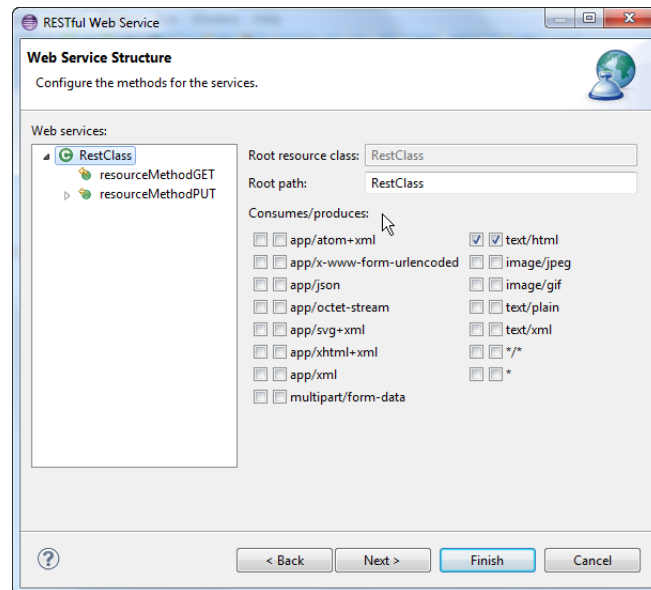
8.3.1 How to Create a Patterned REST Web Service

There are three options to choose from for your patterned REST web services:

- **Single Source**, - Creates a root-level JAX-RS resource class with GET and PUT methods.
- **Resource/sub-resources** - Creates a pair of JAX-RS resource classes. The 'item' class represents individual resources in a collection, the other class is for the container that houses the collection. If the item resource URI is determined by the client, the item class is created using the PUT method in the item class, instead of the POST method.
- **Entities Access** - Creates JAX-RS compliant web services from JPA entity classes.

To create a Single Source REST web service using the wizard:

1. In the OEPE application choose **New > Other > Web Services > REST Web Service**. The Web Service Pattern dialog appears, as shown in [Figure 8-5](#).
2. The dialog has three REST pattern types from which to choose. Choose the Single Source pattern type and click **Next**, if you want to configure the REST methods, or click **Finish** to use the defaults.
3. If you chose Next, the Web Service Structure dialog appears with the REST Java class and the methods, as shown in [Figure 8-7](#). You can accept the root path default or change it. Under Consumes/produces, choose the format to use to when your REST methods are consumed or produced for the root resource class. For more information see, [Section 8.5, "Customizing Media Types for the Request and Response Messages"](#).
4. Click **Next**. The Rest Application Class dialog appears. Choose to accept the default, or specify the Package and Class name added to the REST configuration, which is used at runtime to locate all the web services in the project.

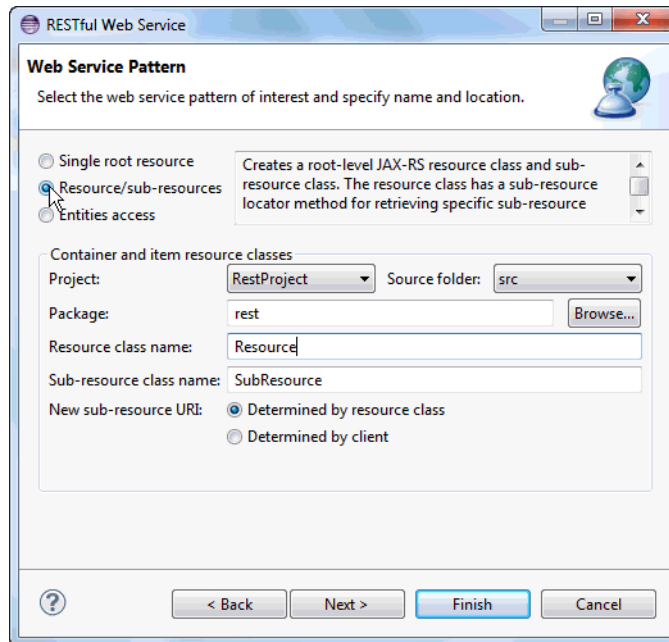
Figure 8–5 REST Create Patterned Web Service - Single Root Resource**Figure 8–6** REST Create Patterned Web Service - Configure Methods

To create a Resource/Sub-Resources REST web service:

1. In the OEPE application choose **New > Other > Web Services > REST Web Service**. The Web Service Pattern dialog appears, as shown in [Figure 8–7](#).
2. The dialog has three REST pattern types from which to choose. Choose the Resource/sub-resources pattern type and click **Next**, to configure the REST methods, or **Finish** to use the defaults.
3. If you chose Next, the Web Service Structure dialog appears with the REST Java class and the methods, as shown in [Figure 8–7](#). You can accept the root path default or change it. Under Consumes/produces, choose the format in which you want your REST methods to be consumed or produced for the root resource class. For more information see, [Section 8.5, "Customizing Media Types for the Request](#)

and Response Messages".

Figure 8–7 REST Create Patterned Web Service - Resource/sub-resources



To create an Entities Access REST web service

1. In the OEPE application choose **New > Other > REST Web Service**. The Web Service Pattern dialog appears.
2. The dialog has three REST pattern types from which to choose. Choose the Entities Access pattern type and click **Next**, if you want to configure the REST methods, or **Finish** to use the defaults.
3. If you chose Next, the Web Service Structure dialog appears with the REST Java class and the methods, as shown in [Figure 8–7](#). You can accept the root path default or change it. Under Consumes/produces, choose the format in which you want your REST methods to be consumed or produced for the root resource class. For more information see, [Section 8.5, "Customizing Media Types for the Request and Response Messages"](#).

8.3.2 How to Create a POJO REST Web Service

Once your project is configured for REST (see [Section 8.2, "Creating Projects Configured for REST"](#)), you can use annotations on your POJOs to define your REST web service URIs. You can either do this directly on the file, or using the Rest AnnotationProperties window. The `@Path` annotation defines the relative URI path for the resource, and can be defined as a constant or variable value (referred to as "URI path template"). You can add the `@Path` annotation at the class or method level.

To create REST web services in the Java source editor:

To define the URI as a constant value, pass a constant value to the `@Path` annotation. Preceding and ending slashes (/) are optional, as shown in [Example 8–3](#).

Example 8–3 Using Annotations to Define the Resource Class URI as a Constant

```
package samples.helloworld;
```

```
import javax.ws.rs.Path;
...
// Specifies the path to the RESTful service
@Path("/helloworld")
public class helloWorld { . . . }
```

In [Example 8-4](#) shows the relative URI for the resource class defined using a variable, enclosed in braces.

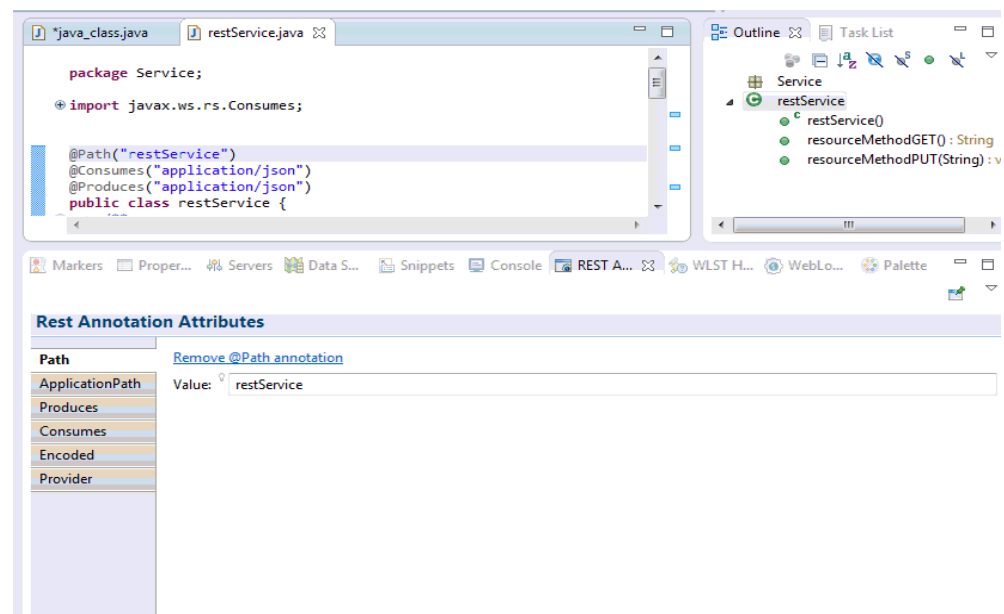
Example 8-4 Using Annotations to Define the Resource Class URI as a Variable

```
package samples.helloworld;
import javax.ws.rs.Path;
...
// Specifies the path to the RESTful service
@Path("/users/{username}")
public class helloWorld { . . . }
}
```

To create REST web services using the Annotation view

1. Create a POJO.
2. Open the Annotations view.
3. Put Cursor on the type declaration, this will orient the available annotations to this location.
4. Add the `@Path` Annotation, specify the value of your path, which is the URI for your resource class. This enable this class as a REST resource, as shown in [Figure 8-8](#).
5. Select applicable methods and annotate with the appropriate `@Path`, `@Get`, `@Put`, `@Post`, `@Delete`.

Figure 8-8 Add REST to POJO - Annotations View



8.4 Mapping Incoming Requests to Java Methods

JAX-RS uses Java annotations to map an incoming HTTP request to a Java method. The process for mapping HTTP Requests/Responses will be a function of the MediaTypes designated by the consumes/produces.

For more information on annotations and mapping, see Mapping Incoming HTTP Requests to Java Methods, in *Developing and Securing RESTful Web Services for Oracle WebLogic Server*.

8.4.1 How to Map an HTTP Request to Java Methods in the REST Generation Wizard

A request method designator annotations are runtime annotations, defined by JAX-RS, and which correspond to the similarly named HTTP methods. Within a resource class file, HTTP methods are mapped to Java programming language methods using the request method designator annotations. The behavior of a resource is determined by which of the HTTP methods the resource is responding to. Jersey defines a set of request method designators for the common HTTP methods: @GET, @POST, @PUT, @DELETE, @HEAD.

To map an HTTP Request to a Java method in the REST wizard

These steps assume that you have a JPA faceted project with generated entities. You will also need to have the JAX-RS facet enabled. (See [Chapter 8.2, "Creating Projects Configured for REST"](#).)

1. Select **File > New > Other > Web Services > Rest Web Services**.
2. Select **Entities**. The HTTP Type is designated given the following rules:
 - PUT- method to create or update a storage container.
 - GET - Requests data from a specified resource.
 - POST-Submits data to be processed to a specified resource.
 - DELETE-Deletes the specified resource.
 - HEAD-Same as GET but returns only HTTP headers and no document body.
3. Open your POJO in the source editor.
4. Open the REST Annotations View.
5. Place your cursor on the Class Type and Specify the @Path Annotation, making this a JAX-RS.
6. Place your cursor on the given method and add the appropriate JAX-RS request method designator annotation as defined by the HTTP type rules.

8.4.2 How to Map HTTP Requests to Java Methods in the Java Class

The `javax.ws.rs.GET` annotation transmits a representation of the resource identified by the URI to the client. The format or the representation returned in the response entity body can be HTML, plain text, JPEG, or another form. [Example 8-5](#) shows how to map an HTTP GET Request to a Java method in a class called `BookmarksResource`.

Example 8-5 Mapping the HTTP GET Request to a Java Method

```
import javax.ws.rs.GET;
import javax.ws.rs.Produces;
import javax.ws.rs.Path;
...
```

```

public class BookmarksResource {
    ...
    @Path("{bmid: .+}")
    public BookmarkResource getBookmark(@PathParam("bmid") String bmid) {
        return new BookmarkResource(uriInfo, em,
            userResource.getUserEntity(), bmid);
    }
    @GET
    @Produces("application/json")
    public JSONArray getBookmarksAsJsonArray() {
        JSONArray uriArray = new JSONArray();
        for (BookmarkEntity bookmarkEntity : getBookmarks()) {
            UriBuilder ub = uriInfo.getAbsolutePathBuilder();
            URI bookmarkUri = ub .
                path(bookmarkEntity.getBookmarkEntityPK().getBmid()).
                build();
            uriArray.put(bookmarkUri.toASCIIString());
        }
        return uriArray;
    }
    ...
}

```

8.4.3 How to Map HTTP Requests to Java Methods in the Annotations View

Use the Annotations view to add your request and response requests to your POJOs.

To create REST web services using the Annotation view

1. Create a POJO.
2. Open the Annotations view.
3. Put cursor on the type declaration, this will orient the available annotations to this location.
4. For the request customization, add the `@Consumes` annotation and specify the value. This sets the value of the media type for that annotation. Note that when `@Consumes` is applied at the method level, it overrides any `@Consumes` annotations applied at the class level.

For the response customization, select the `@Produces` annotation in the properties view. Select the appropriate media type for your response. Note: If it is applied at the method level, it overrides any `@Produces` annotations applied at the class level.

8.5 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.

With OEPE, you can customize message types for request and response using one of the following options:

- [How to Create a Patterned REST Web Service](#)
- [How to Customize Media Types in the Java Source Editor](#)
- [How to Customize Media Types in the Annotations View for a Java Class](#)

8.5.1 How to Customize Media Types in the Java Source Editor

Customize media types for your REST web service by adding the `@Produces` or `@Consumes` annotation at the class or method level of the resource. Then you can input the media types values for your produces and consumes attributes, as shown in [Example 8–6](#).

Example 8–6 Customizing the Media Types for the Response using `@Produces` Annotation

```
package samples.produces;
import javax.ws.rs.Produces;
import javax.ws.rs.Path;

@Path("/myResource")
@Produces("text/plain")
public class GetDogTreats {
    @GET
    public String doGetAsPlainText() { ... }
    @GET
    @Produces("text/html")
    public String doGetAsHtml() { ... }
}
```

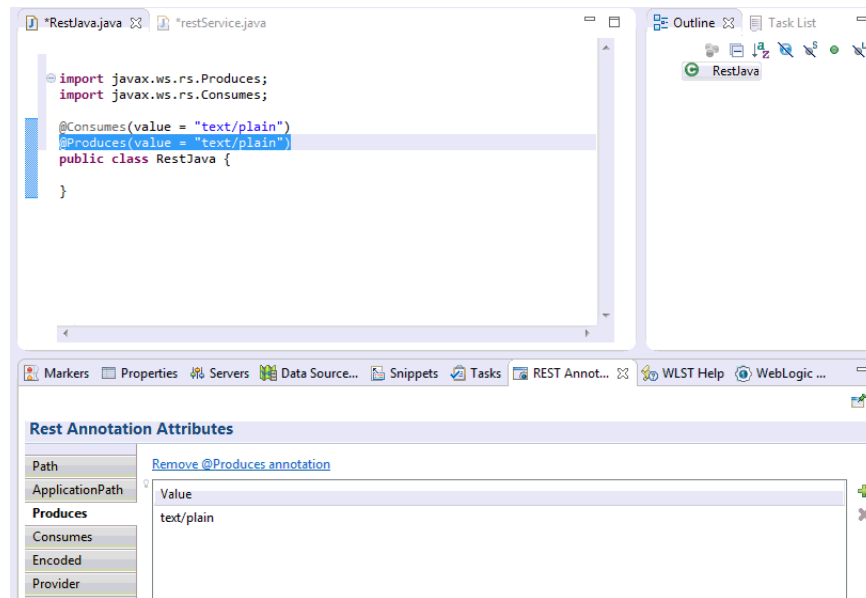
8.5.2 How to Customize Media Types in the Annotations View for a Java Class

Use the Annotations view to add your media type annotations and properties to your Java classes.

Note: Explicit use of the `javax.ws.rs.core.MediaType` String constants is not directly supported through the annotations view.

To create REST web services using the Annotations view

1. Create a POJO.
2. Open the Annotation view.
3. Put cursor on the type declaration, this will orient the available annotations to this location.
4. Add the `@Consumer` or the `@Produces` annotation, and specify the value of your annotation.
5. Specify the resource value of your annotation. Choose from the available values for that annotation. For example `@Produces("text/plain")`, as shown in [Figure 8–9](#).

Figure 8–9 Adding Media Type REST Annotations in the Annotations View

8.6 Validation and Quick Fix

REST services are validated based on the REST annotations present at the class, method, and method parameters level. The validation is done in the REST annotation processor.

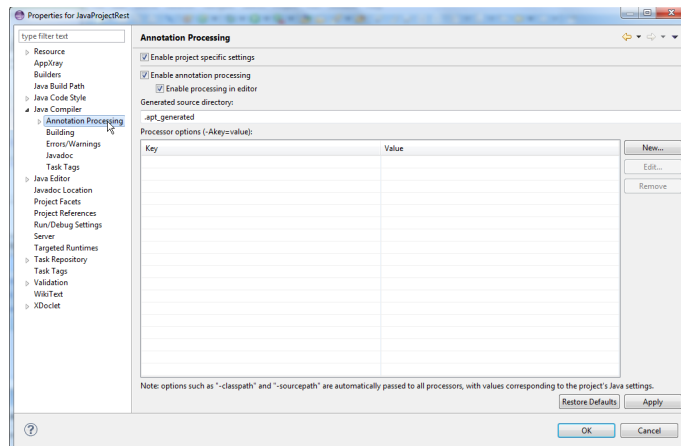
To see the validation messages on a REST service, turn on annotation processing at the project level, through the project properties dialog. By default, annotation processing on a project is not enabled.

There is a quick fix bulb icon on the validation output window which you can select to resolve the following:

- Issues with the values in the PathParam annotation.
- Issues related to converting a non-public resource method to a public method.

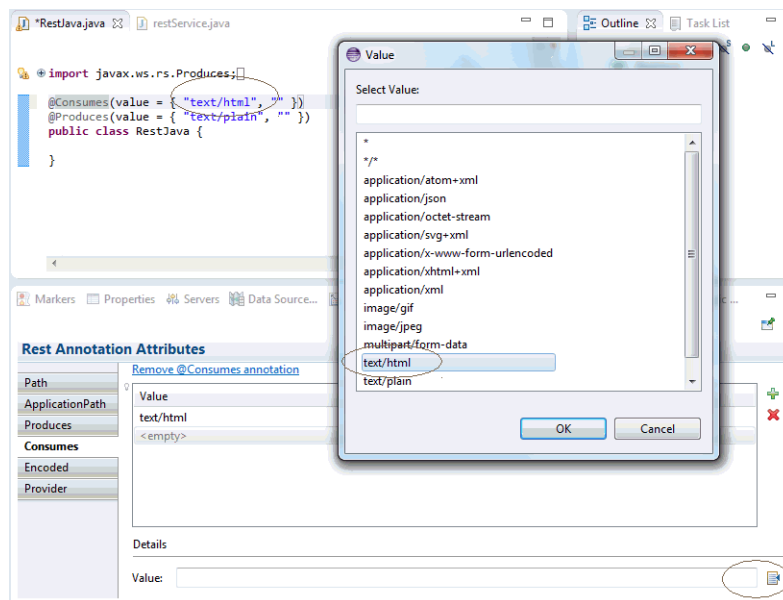
To show annotation validations:

1. Go to **File > Properties**.
2. Select **Java Compiler > Annotation Processing**.
3. Select **Enable project specific settings**, and **Enable annotation processing**.

Figure 8–10 Properties - Annotations

8.7 Content Assist

For annotations that require values, the Produces and Consumers annotations, for example, you can use Content Assist to show you the available options and choose your value. To use the Content Assist, from the annotations view click the Add symbol. Then select the <empty> field. This enables the value field below under Details. Click the page icon. This brings up a list of available values for that annotations shown in [Figure 8–11](#)

Figure 8–11 Content Assist Annotations Values

8.8 Run-AS JAX-RS Support

There are two types of server scenarios to run your REST projects.

- **Run on Server** - deploys the REST application to the Project Targeted Runtime J2EE Server.

- Run on a **JAX-RS Resource Application** - deploys to a lightweight built-in HTTP Server that designed to quickly deploy a simple JAX-RS resource.

Note: When you change the class or project class path in the run configuration wizard, you need to manually stop the JAXRS application and start to persist changes. Unless it is stopped and started again, it will not reflect the changes.

8.8.1 How to Deploy to a Targeted Runtime J2EE Server

To use the run-as feature, your project needs to have an implementation class annotated with `@Path` that is packaged with a servlet specified in the `web.xml` deployment descriptor.

To Run-As > Run On Server to deploy to a targeted J2EE runtime:

Navigate to the implementation class, right-click and choose **Run-as > Run On Server**. This publishes the application to the server. Once successfully deployed, the Browser is launched with a url specified to target the `application.wadl` that corresponds to this packaged resource.

The targeted browser can be controlled through the Eclipse Workbench by selecting **Window > Preferences > General > Web Browser**.

Note: Advanced packaging through the application subclass is not supported, as well as the `@Path` REST resources specified on super types.

8.8.2 How to Deploy to a Basic HTTP Lightweight Server

Use the **Run-As > JAX-RS Resource** feature to run a basic HTTP Server.

Note: The REST web service is a lightweight HTTP Server, used for running simple REST compliant POJOs. It is not a fully compliant J2EE server. For complex JAXB, JPA, EJB implementations please defer to the project target J2EE runtime container, per the steps in "[How to Deploy to a Targeted Runtime J2EE Server](#)"

To Run-AS > JAX-RS Resource Application

On your JAX-RS annotated simple POJO, right-click and select Run-As > JAX-RS Resource Application. This will launch the Jersey HTTP Server, and by default will direct the browser launch to the `application.wadl`. By default it uses port 8080, if this port is already in use, you can define another port by doing the following.

1. Right-click your POJO again, and choose **Run As > Run Configurations**.
2. Select the JAX-RS Application with the corresponding class name, as shown in [Figure 8-12](#). Under the Query URL, add your port configuration, as shown in [Figure 8-13](#).
3. In the JAX-RS Application Run Configuration dialog, create a new configuration by selecting the project and JAX-RS annotated resource. Use the browse and search options to find your available resources.

4. Change the default Query Url options to customize the browser target url launch behavior. Use the additional tabs to customize the Java runtime behavior, including the VM arguments, environment variables, JRE, classpath, and common.

Figure 8–12 Run Configuration - Resources

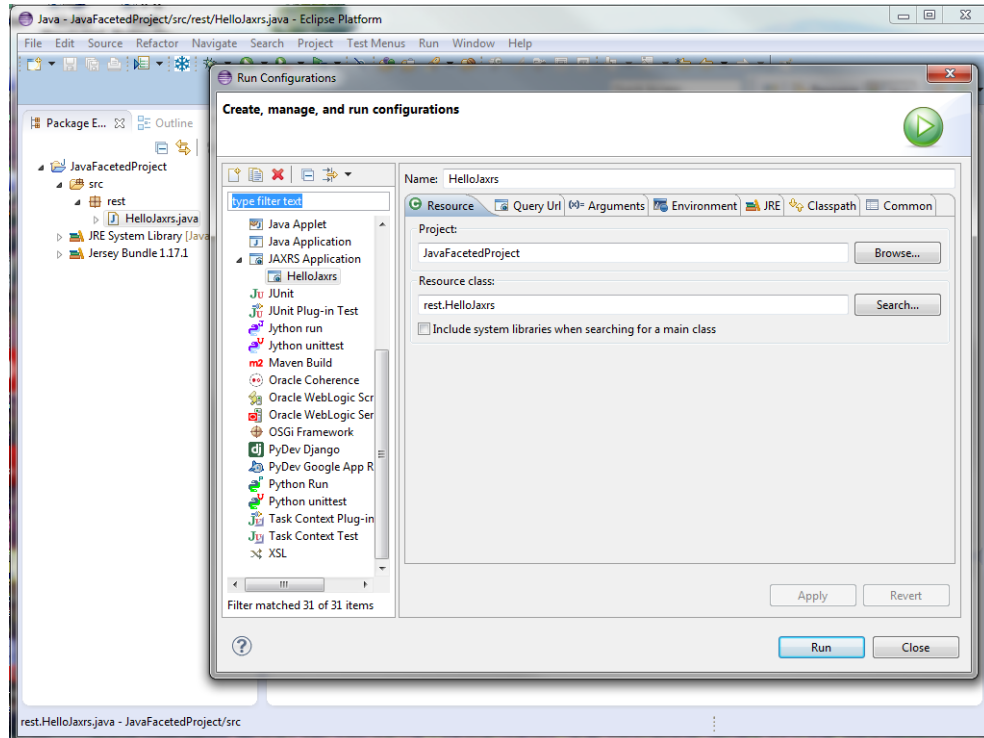
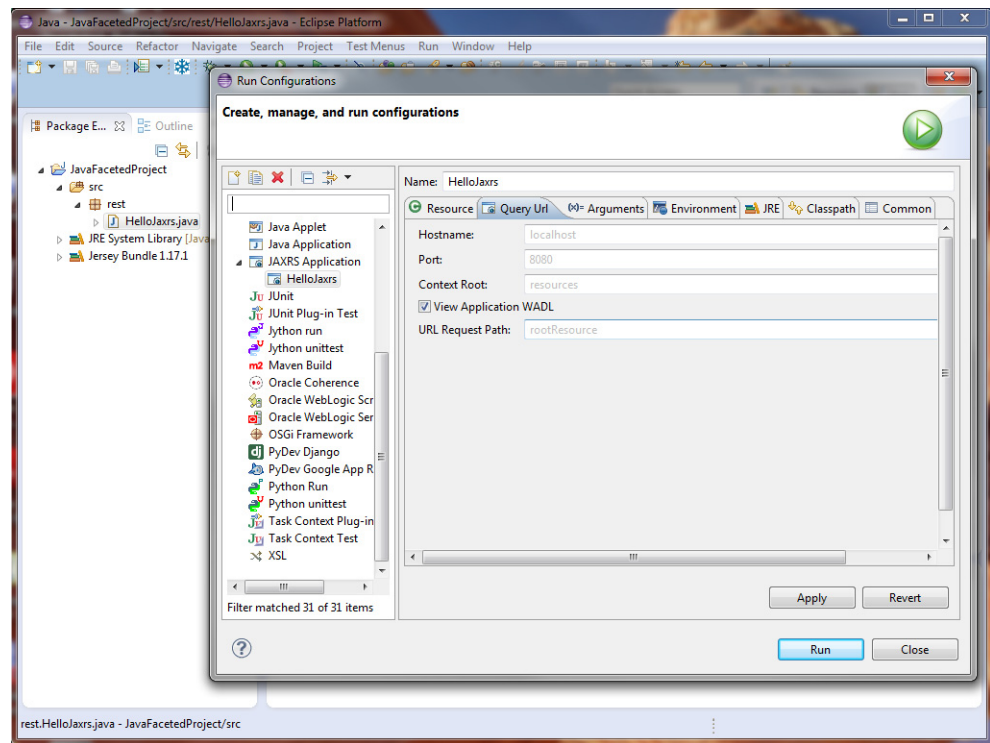


Figure 8–13 Run Configuration - Query URL



8.9 Generate a Java REST Client from a WADL

The Web Application Description Language (WADL) is an XML-based file format that describes your REST web services application. By default, a basic WADL is generated at runtime and can be accessed from your REST web service by adding a GET to the `/application.wadl` resource at the base URI of your REST application. For example: GET `http://<path_to_REST_app>/application.wadl`.

You can also use the options method to return the WADL for particular resource, as shown in [Example 8–7](#).

Example 8–7 WADL Describing a REST Web Service

```
<?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: 0.10-ea-SNAPSHOT 08/27/2008 08:24 PM"/>
  <resources base="http://localhost:9998/">
    <resource path="/helloworld">
      <method name="GET" id="sayHello">
        <response>
          <representation mediaType="text/plain"/>
        </response>
      </method>
    </resource>
  </resources>
</application>
```

Oracle Database Support

The OEPE database support lets you easily connect to, create, explore, and query Oracle databases. Support includes a graphic editor for SQL schemas, and DDL generation.

This document contains the following sections:

- [Getting Started with the Oracle Database Plugin for Eclipse](#)
- [Using the RDB Schema Editor](#)

9.1 Getting Started with the Oracle Database Plugin for Eclipse

Welcome to the first step towards supporting the Eclipse Data Tool Platform (DTP) by Oracle. This document provides a high-level tour through each of the Oracle Database Plugin features.

9.1.1 Using the Database Explorer

The Database Explorer is the Datasource View provided by the DTP. You use it to create Database connections and to navigate the database.

9.1.1.1 Creating a Connection to a Database

You can create a connection to a database.

To create a connection to a database using the Data Source Explorer view:

1. To open the Database Development perspective, click **Windows > Open Perspective > Other** from the Main menu, and then select **Database Development** from the Open Perspective dialog. This perspective opens the Data Source Explorer (DSE) view.
2. Right click on the Databases node in DSE and select **New** to create a database connection.
3. Select **Oracle Database Connection** from the list and provide a name for the connection. Click **Next** to proceed to the next step.
4. Complete the rest of the dialog as follows:
 - Select either Oracle Database 10g Driver Default or Oracle Database 11g Driver Default from the drop-down list of drivers.
 - If the default settings are not appropriate for your configuration, change the following information on the Properties > General tab of the New Connection Profile dialog:

- In the **SID** field, replace `xe` with the Service Name (or SID) of the Database Service.
 - In the **Host** field, replace `server` with the hostname or IP address of the Oracle Database server (possibly `localhost`).
 - In the Port number field, replace `1521` with the port number of the Oracle Database Listener service.
 - Provide the database user name and password for the connection.
Note that the Connection URL field reflects your changes.
 - Select a use group in the Catalog field.
 - On the Properties > Optional tab, set the following optional properties, if required:
 - `autocommit=false`
By default, `autocommit` is set to `true`, which results in immediate commit of the operations from SQL Editor. When set to `false`, you need to execute the explicit `COMMIT` to commit the changes. See Oracle Database SQL Reference for more information.
 - `sysdba=true`
Use this property to login with `SYSDBA` account.
 - `sysoper=true`
Use this property to login with `SYSOPER` account.
 - Click **Test Connection** to test the connectivity. If this fails, try restarting Eclipse with `-clean` command option.
 - Select **Connect** when the wizard completes checkbox to enable the database connection.
5. Select **Finish** to complete the wizard.

9.1.1.2 Working with a Database Connection

The open database connection allows you to navigate through the database objects.

9.1.1.3 Editing Data in a Table

With the enabled database connection, you can edit the table data in the Data Source Explorer (DSE). Note that, for comprehensive editing capabilities, it is recommended that you install Data Tools Platform SQL Development Tools Data Functions feature.

To install the Data Tools Platform SQL Development Tools Data Functions feature:

1. If your machine is located inside of a network, which requires a proxy to access outside resource such as the Internet, the configuration (download) may fail due to the fact that Eclipse IDE includes a Web browser to let you access the Internet from within the IDE. In this case, reconfigure your Eclipse IDE proxy settings using **Window > Preferences > General > Network Connections**.
2. Select **Help > Software Updates** from the main menu.
3. Open the Available Software tab on the Software Updates and Add-ons dialog.
4. Click **Add Site** to install the feature provided by OEPE from OEPE's update site. To do so, enter

http://www.oracle.com/technology/software/products/oepe/oepe_12g.html in the **Location** field of the Add Site dialog, and then click **OK**.

5. Expand the Ganymede Update Site node, and then expand the Database Development node. Select **Data Tools Platform SQL Development Tools Data Functions**, and then click **Install**.

To edit a table data:

1. Navigate to the table you want to edit in the DSE, then right-click the table, and select **Data > Edit**. This opens the table data in the editor.
2. Make changes to the table data by right-clicking on a table cell and using the popup menu. When you have finished editing, click **Save** to save the changes to the database.

9.1.1.4 Loading Data into a Table

You can load data from a text file into a table using the DSE.

To load data into a table:

1. In the DSE, navigate to the table into which you want to load data. Right-click the table, and select **Data > Load** from the drop-down menu. This opens the Load Data dialog.
2. Complete fields on the dialog, and then click **Finish**. Validation is performed before the data is loaded.

9.1.1.5 Extracting Data from a Table

To extract data from a table to a text file using the DSE.

To extract the data:

1. In the DSE, navigate to the table from which you want to extract data. Right-click the table and select **Data > Extract** from the drop-down menu. This opens the Extract Data dialog, as Figure 11 shows. Complete fields on the dialog, and then click **Finish**.
2. Complete fields on the dialog, and then click **Finish**.

9.1.1.6 Generating DDL

You can use the Generate DDL option on most database objects to create or drop the object.

To generate DDL:

1. If necessary, create a project in order to save the generated DDL Script.
2. In the DSE, navigate to the object you want to create or drop, right-click the object, and select **Generate DDL** from the drop-down menu to create a DDL script.

The DDL is generated.

9.1.2 SQL Tools

SQL Tools enable you to edit and run stored procedures and functions, as well as execute the so-called explain plans in either graphic or text mode.

9.1.2.1 Using SQL Editor

The SQL Editor enables standard text-based editing of SQL statements, provides syntax color, and multiple statement support.

To use a SQL Editor:

1. In the DSE, navigate to the procedure or function you want to edit.
2. Right-click the procedure or function and select **Edit** from the drop-down menu. The procedure or function opens in the SQL Editor.

9.1.2.2 Executing a Stored Procedure or Function

You can execute stored procedures and functions.

To execute a stored procedure or function:

1. In the DSE, navigate to the procedure or function you want to run.
2. Right-click the procedure or function and select **Run** from the drop-down menu.
3. If the procedure or function has any input parameters, the Configure Parameters dialog appears.
Enter input values and click **OK** to run the procedure or function.

9.1.2.3 Executing Explain Plans

You can use explain plans to optimize your code.

To execute the explain plan:

1. In the Navigator or DSE, navigate to the script containing the SQL statement for which you want to execute an explain plan.
2. Right-click on the script and select either **Execute Text Explain Plan** or **Execute Graphic Explain Plan** from the drop-down menu.

For example, open the `views.sql` file that you created in the Generating DDL section. Highlight the `SELECT` statement block. Right-click and select **Execute Graphic Explain Plan** from the drop-down menu. This opens the execution plan in graphic mode in the Execution Plan view.

Alternatively, if you select **Execute Text Explain Plan** from the drop-down menu, it will result in a text version of the execution plan.

9.1.3 Granting and Revoking Privileges

To grant specific database privileges to a specific user:

1. In the DSE, navigate to the element (such as a table, for example) for which you want the user to have certain privileges.
2. Right-click the element and select **Grant Privileges** from the drop-down menu. This opens the Grant Privileges dialog.
3. Select one of the privileges from the list and click **OK**.

To revoke specific database privileges from a specific user:

1. In the DSE, navigate to the element (such as a table, for example) for which you want to revoke the user privileges.
2. Right-click the element and select **Revoke Privileges** from the drop-down menu. This opens the Revoke Privileges dialog.

3. Select one of the privileges from the list and click **OK**.

9.1.4 Creating Tables

Using the tool, you can create new database tables by declaring new columns, defining primary, unique, and foreign keys, as well as adding checked constraints and indices.

To create a new database table, follow this procedure:

1. In the DSE, navigate to the Tables element.
2. Right-click the element and select **New Table** from the drop-down menu. This opens the Create Table dialog.

Even though by default the table is created using the schema of the user who established the connection, you can change the database schema. If the user who established the connection does not have privileges to create a new table in another schema, the SQL Result view will display an error message after the wizard completes.

3. Using the Create Table dialog, specify the following:
 - The name for your table.
 - One or more table columns: to define a column, click **Add**, and then provide the column's name, select the type using the Type dialog, and specify whether or not the value can be `null`.

You can reorder the table columns using the Move Up and Move Down buttons.

- Click **Next** on the Create Table dialog to open the Create Table > Primary Key dialog. On this dialog, select Add primary key, and then define the name and select one or more columns that make up the primary key. If a column type is not suitable as a primary key column, a validation error will be displayed. Proceed by clicking **Next**.
- On the next Create Table > Unique Constraints dialog, define unique constraints for the table by clicking **Add**. When defined, click **Next** to proceed.

There are three types of validation for the unique constraints:

- Each column must be suitable for unique constraints. For example, a column of type CLOB is not suitable as a column in unique constraints.
- A unique constraint must not contain the same columns as the primary key.
- A unique constraint must not contain the same columns as another unique constraint.
- On the next Create Table > Foreign Keys dialog, define foreign keys for the table by clicking **Add**.

Note: Foreign keys can reference the primary key of a table which resides in another schema. You can switch schemas by clicking **Browse** for the **Referenced schema** field. By clicking **Browse** for the **Referenced table field** you can switch to different tables in the referenced schema. When the referenced table is selected, the wizard will look up its primary key and fill the **Referenced constraint** field if the primary key exist. It will also try to match the columns in the referenced table with columns in the new table based on their types. Matched columns will show in the Associations table. If there are multiple columns that match a column in the referenced table, the wizard will choose the first one it finds, which you can change.

4. When finished, click **Next**.
5. Using the next Create Table > Check Constraints dialog, add checked constraints for the new table by clicking **Add**.

Note: The Condition field is represented by a free-formatted text area. When using it, you have to ensure that the text for the checked constraints conforms to PL/SQL syntax. Otherwise, you will not be able to set the condition successfully.

6. When finished, click **Next**.
7. Use the next Create Table > Indices dialog to add create indices for the new table by clicking **Add**, and then click **Next**.
8. The next Create Table > DDL dialog shows summarizes the DDL used to create the new table. You can make changes to the DDL before it is executed. Note that if you do make changes to the DDL, and then go back to previous dialogs and decide to add a new column, then your changes to the DDL will be lost and a new DDL will be generated.
9. Click **Finish** to complete the wizard.

Upon completion, the table creation DDL is sent to the JDBC driver and executed there. If the execution is successful, a new table with all the specified constraints will be created. You can examine the result of new table creation from the SQL Result view that opens.

Note: The table creation DDL is not executed as a single SQL statement. For any constraint, an `ALTER TABLE` statement is executed. If any of the `ALTER TABLE` statements fails, you have the option on whether or not to continue to execute subsequent statements.

If the newly created table is not displayed in the Tables node in Data Source Explorer, you will need to refresh the view.

9.1.5 Troubleshooting

Unable to sort folders

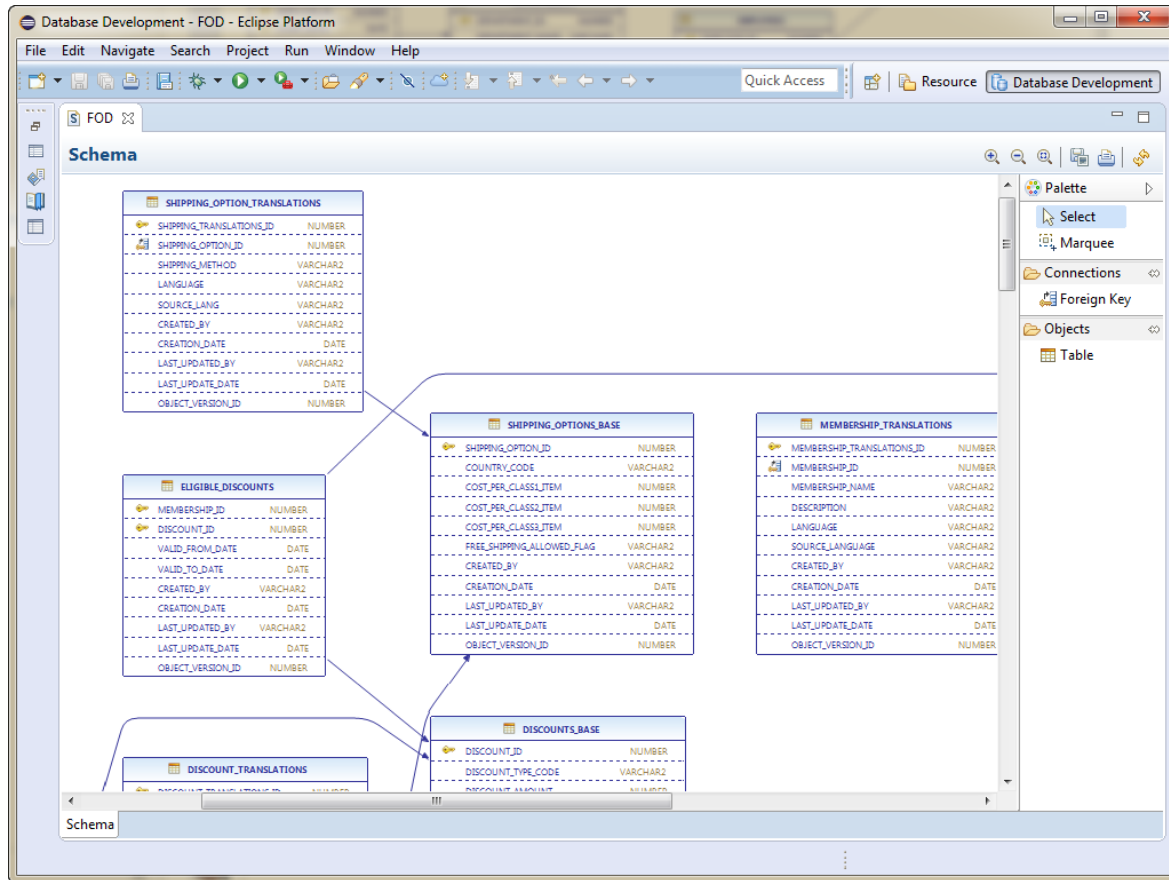
This issue results in an inability to sort the nodes on Data Source Explorer (DSE) in an order that is consistent with how it is displayed in Oracle SQL Developer.

9.2 Using the RDB Schema Editor

OEPE allows you to examine and edit your database schema using the RDB Schema Editor that displays tables and the relationships between them, as shown in [Figure 9-1](#).

The editor displays tables as table nodes. Each node lists all the columns in a table and shows column data types. The node also provides primary and foreign key indicators in a form of icons. Foreign key relationships between tables are represented by links in a form of arrows.

Figure 9-1 Database Objects in RDB Schema Editor



9.2.1 How to Display a Database Schema in the Editor

Before you can use the RDB Schema Editor you must set up and configure a database connection. For more information, see [Section 9.1.1.1, "Creating a Connection to a Database."](#)

You can open the editor from:

- A schema
- A table in the schema. In this case, the table is selected in the editor when it opens.

To open the RDB Schema Editor:

1. Choose **Window > Show View > Data Source Explorer** from the top-level menu. This opens the Data Source Explorer view.

2. Expand the database connection node to the database, and then to the schema or a particular table in the schema.
3. Right-click on the schema or table name, and choose **edit**. OEPE builds a graphical model for the database schema and opens the editor to display the tables in the schema and their relationships.

9.2.2 Working with RDB Schema Editor Features

The schema editor gives you a visual view of your database schema and the relationships between the tables. In the editor, you can:

- Manipulate the layout of the schema in the editor. Select one table using the Select tool in the Palette, or a group of tables by selecting the Marquee tool in the Palette and drawing around the tables you want.

Move tables by selecting one or more of them and dragging to a new position. Move foreign keys by clicking on the line to create a bend point, then drag the bend point to the new location.


When the editor is saved, the layout is also saved so that the next time the editor is launched from the same schema the saved layout is used.

- When you select a single table some editing tools become available, as shown in [Figure 9–2](#).

Figure 9–2 Table Editing Tools

SHIPING_OPTIONS_BASE	
SHIPING_OPTION_ID	NUMBER
COUNTRY_CODE	VARCHAR2
COST_PER_CLASS_ITEM	NUMBER
COST_PER_CLASS2_ITEM	NUMBER
COST_PER_CLASS3_ITEM	NUMBER
FREE_SHIPPING_ALLOWED_FLAG	VARCHAR2
CREATED_BY	VARCHAR2
CREATION_DATE	DATE
LAST_UPDATED_BY	VARCHAR2
LAST_UPDATE_DATE	DATE
OBJECT_VERSION_ID	NUMBER

- To add a column, click + and edit the new column entry in the table.
- To delete a column, select it and click X.
- To show the table properties, click Show in Properties View. The Properties window, which opens by default under the schema editor, displays detailed information about the selected table, column or foreign key. You can edit any fields in the Properties window.
- To save changes to the table, click Save.
- Edit table names, column names, column types and other column constraints such as size and scale. When the editor is saved, all the tables on the diagram are compared against the database. Where there have been changes to tables, the database version of the table is dropped and recreated using the version in the editor.
- Edit table names, column names, column types and other column constraints such as size and scale. When the editor is saved, all the tables on the diagram are compared against the database. If a table is found to be different than what is in the database, either:

- An ALTER TABLE statement is used to update the table in the database.
- If that is not appropriate, for example, when columns are reordered, the table will be dropped and recreated. OEPE will warn you if the table being dropped contains data.
- You can create foreign keys using  (Foreign Key) in the Palette. Click on the source table then on the destination table. The Define Foreign Key wizard opens where you can specify the column mappings. When you click Finish in the wizard, the foreign key is created.
- You can drop foreign keys by choosing the Select tool from the Palette, selecting the foreign key and pressing delete.

The context menu of the editor allows you to perform the following:

- Refresh the schema in the editor.
- Add a new table.
- Select all tables.
- Select all nodes.
- Show a grid.
- Show guides.
- Automatically adjust the layout horizontally or vertically.
- Zoom in, zoom out, or zoom to the actual size.
- Save the layout as a PNG image file.
- Send the layout to a printer.

Object-Relational Mappings Support

Object-Relational Mapping Tools are designed to help develop, deploy, and debug ORM JPA applications for Oracle WebLogic Server.

This document contains the following sections about configuring the persistence provider for JPA projects:

- [Configuring a JPA Project to Use EclipseLink Persistence Provider](#)
- [Configuring a JPA Project to Use Kodo Persistence Provider](#)
- [Oracle WebLogic Server Support for Persistence Provider Libraries and Deployment](#)

10.1 Configuring a JPA Project to Use EclipseLink Persistence Provider

EclipseLink is an open source persistence provider contributed to Eclipse by Oracle.

For more information, see the "EclipseLink User's Guide - Developing JPA Projects" which is available on the Eclipse site at http://wiki.eclipse.org/EclipseLink/UserGuide/Developing_JPA_Projects_%28ELUG%29.

OEPE provides an EclipseLink project facet that you can use in your Eclipse JPA project. When selected, the following happens:

- The project build path is automatically configured to include EclipseLink persistence provider JAR files shipped with Oracle WebLogic Server 11gR1.

Note: Even though the library files are not shipped with earlier version of Oracle WebLogic Server, you can download them using the facet configuration wizard. For more information, see [Section 10.3, "Oracle WebLogic Server Support for Persistence Provider Libraries and Deployment."](#)

- The database connection properties specific to EclipseLink can be automatically configured in the `persistence.xml` file of your Eclipse JPA project.

Oracle WebLogic Server releases prior to 11gR1, as well as third-party J2EE application servers, support EclipseLink persistence provider through the user library provider mechanism. For more information, see [Section 10.3, "Oracle WebLogic Server Support for Persistence Provider Libraries and Deployment."](#)

To configure your Eclipse JPA project to use EclipseLink as a persistence provider, use one of the methods described below.

To automatically configure your JPA project to use EclipseLink persistence provider shipped with Oracle WebLogic Server 11gR1:

1. If necessary, create a new JPA project. From the main menu, select **File > New > JPA Project**.
In the New JPA Project dialog, choose the Oracle WebLogic Server 11gR1 as the **Target Runtime** and complete the details on the New Server Runtime Environment dialog.
2. Click **Modify** in the Configuration portion on the New JPA Project > JPA Project dialog. This opens the Project Facets dialog.
3. Select **Java Persistence Library (EclipseLink)** from the Project Facet list, and then click **OK** to apply your selection and close the Project Facets dialog.
4. Click **Next** on the New JPA Project > JPA Project dialog.
5. On the next, JPA Facet screen of the New JPA Project dialog, select **EclipseLink** as the platform for your project.
6. Specify the database connection in the **Connection** field.
7. In the JPA implementation portion of the New JPA Project dialog, select **Use implementation provided by server runtime**, and then click **Next**. This opens the New JPA Project > EclipseLink screen.
8. Select **EclipseLink version-number**, and then click **Finish**. Note that if the EclipseLink library is not listed, you need to click **Download library**. If your machine is located inside of a network, which requires a proxy to access outside resource such as the Internet, the download may fail due to the fact that Eclipse IDE includes a Web browser to let you access the Internet from within the IDE. In this case, reconfigure your Eclipse IDE proxy settings using **Window > Preferences > General > Network Connections**, and try again.

If you expand your JPA project node in the Project Explorer, and then expand the EclipseLink node, you will find the EclipseLink persistence provider library, as well as Java persistence library.

The `persistence.xml` file of your JPA project now contains EclipseLink persistence provider settings.

To manually configure a Java user library which includes the EclipseLink JAR files that you downloaded separately:

1. When creating a new JPA project, select **Oracle WebLogic Server 11gR1** as your target runtime.
2. Click **Modify** in the Configuration portion on the New JPA Project > JPA Project dialog. This opens the Project Facets dialog.
3. Select **Java Persistence Library (EclipseLink)** from the Project Facet list, and then click **OK** to apply your selection and close the Project Facets dialog.
4. On the next, JPA Facet screen of the New JPA Project dialog, select **EclipseLink** as the platform for your project, and then specify the database connection in the **Connection** field.
5. In the JPA implementation portion of the New JPA Project > JPA Facet dialog, leave the default selection of **Use implementation provided by server runtime**, and then click **Next**. This opens the New JPA Project > EclipseLink dialog. Click **Download library**. This opens the Download Library dialog. Select the EclipseLink library you would like to use, and click **Next**. Note that if your machine is located inside of a network, which requires a proxy to access outside

resource such as the Internet, the download may fail due to the fact that Eclipse IDE includes a Web browser to let you access the Internet from within the IDE. In this case, reconfigure your Eclipse IDE proxy settings using **Window > Preferences > General > Network Connections**, and try again.

6. Add the EclipseLink library to your Eclipse JPA project's build path.

10.2 Configuring a JPA Project to Use Kodo Persistence Provider

Oracle Kodo is the default persistence provider for Oracle WebLogic Server 10*n*. OEPE enables automatic configuration and deployment of Eclipse JPA projects that use Kodo as their persistence provider on Oracle WebLogic Server.

OEPE provides Oracle Kodo facet that allows you to configure your JPA project to use Oracle Kodo as the persistence provider. When selected, the following happens:

- The project build path is automatically configured to include Oracle Kodo persistence provider JAR files shipped with Oracle WebLogic Server.
- The database connection properties specific to Oracle Kodo can be automatically configured in the `persistence.xml` file of your Eclipse JPA project.

Note: Earlier versions of Kodo require a build-time class enhancement process. OEPE triggers execution of a Kodo-specific class enhancer on your Kodo JPA project when deployed to Oracle WebLogic Server 10.0. For more information, see [Section 10.3, "Oracle WebLogic Server Support for Persistence Provider Libraries and Deployment."](#)

To configure your Eclipse JPA project to use Oracle Kodo as a persistence provider:

1. When creating a new JPA project, or configuring a Web project or utility with a JPA facet, click **Modify** next to **Configuration** on the New JPA Project dialog. This opens the Project Facets dialog.
2. Select **WebLogic Utility Module Extensions and Java Persistence Library (Oracle Kodo)** from the Project Facet list, and then click **OK** to apply your selection and close the Project Facets dialog.
3. Click **Next** on the New JPA Project > JPA Project dialog.
4. On the next, New JPA Project > JPA Facet screen of the New JPA Project dialog, select **Generic** as the platform for your project.
5. Specify the database connection in the Connection field, and then click **Next**. This opens the New JPA Project > Kodo dialog. Select `Weblogic System Library (Oracle Kodo 4.2)` from the **Library > Type** combo box.
6. Click **Finish**.

If you expand your JPA project node in the Project Explorer, and then expand the WebLogic System Libraries node, you will find the Oracle Kodo persistence provider library, as well as Java persistence library, at the end of the list.

The `persistence.xml` file of your JPA project now contains Oracle Kodo persistence provider settings.

10.3 Oracle WebLogic Server Support for Persistence Provider Libraries and Deployment

Oracle WebLogic Server 12c (12.1.3) supports the EclipseLink persistence provider.

The following table maps the current version of Oracle WebLogic Server and supported persistence libraries.

Table 10–1 Supported Persistence Libraries

Oracle WebLogic Server Version	EclipseLink JPA
12c (12.1.3)	EclipseLink version 2.5.2 and earlier downloadable libraries

Using OEPE, you can generate Spring configuration and beans from persistence mappings.

This document contains the following sections:

- [Generating Spring Artifacts](#)
- [Generating Web Services for Spring Service Beans](#)

11.1 Generating Spring Artifacts

You can use OEPE to automatically generate Spring configuration and beans from persistence mappings.

OEPE supports Spring IDE 2.0 and Spring Framework 2.5.

To generate Spring artifacts, such as the DAO interface and its implementation, the service interface and its implementation, and configuration files, you have to perform the following steps:

- Configure your project for Spring.
- Generate the Spring files and update the configuration.

To configure your project for Spring:

1. Either add the Spring facet when you create a project, or add the facet to an existing project by right-clicking your project in the Project Explorer and selecting **Properties** from the drop-down menu. This opens the Properties dialog.

Note that the Spring facet is available to Web projects, EJB projects and utility projects. Adding the Spring facet to an EJB project requires special attention.

2. In the Properties dialog, select **Project Facets** on the left panel, and then select **Spring** from the Project Facet list.

To configure your project to use Spring Framework 2.5 Runtime library, select the **Spring Framework 2.5 facet**, and then click **Further configuration required**. This opens the Modify Faceted Project > Spring dialog.

Accept the default selections and click **OK**.

If you select either the Spring Framework 2.0 or 2.5 facet on the Properties dialog, and you wish to either use your own Spring runtime installation or download a copy of Spring libraries hosted by Oracle or SpringSource, select **User Library** in the **Library Type** field of the Modify Faceted Project > Spring dialog.

You can also click the **Manage User Library** icon to select a Java User Library pointing a proper local Spring runtime installation.

3. Click **Download Library** icon to indicate your intention to obtain Spring libraries. This will open the Download Library dialog.
4. To use this dialog to download the Spring libraries, make appropriate selections, and then click **Finish** to trigger the download.
5. Optionally, you may click **Manage libraries** icon to open the Preferences > User Libraries dialog to fine-tune your Spring library configuration. Note that by default, the entire library represented by the `spring.jar` file is included.
6. Click **OK** on the Modify Faceted Project > Spring Facet dialog.
7. By clicking **Apply** > **OK** on the Properties dialog, you will complete the configuration of the Spring Framework library for your project.

If you added the Spring facet to a Web project, the project's `web.xml` file will be updated to include the context listener for loading the Spring context at the Web application's start time, as the following example shows:

```
<context-param>
  <param-name>contextConfigLocation</param-name>
  <param-value>/WEB-INF/applicationContext.xml</param-value>
</context-param>
<listener>

<listener-class>org.springframework.web.context.ContextLoaderListener</listener-cl
ass>
</listener>
```

If you are creating Spring service beans and bean configurations for an EJB project, you should keep in mind that, unlike a Web project, there is no standard way to automatically load Spring bean configuration. In this case, a common practice is to define a Web project to help initialize the application context. Consider the following project set-up:

- [EAR project]
- [EJB with Spring facet]
- [Web project with Spring]

It is assumed that the EJB project includes the Spring service beans and bean configuration under the `META-INF/bean.xml` file. In the Web project, find the default `WEB-INF/applicationContext.xml` file created during the addition of the Spring facet, and add a factory bean looking for other bean configurations in the classpath, as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  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/xsd">
  <bean id="bean.factory"
    class="org.springframework.context.support.ClassPathXmlApplicationContext">
    <constructor-arg>
      <list>
        <value>META-INF/bean.xml</value>
      </list>
    </constructor-arg>
  </bean>
</beans>
```

With this setting, the client code (such as a servlet, for example) in the JSP can access the Spring beans residing in the EJB project. Consider the following code sample from a JSF Web application:

```

ApplicationContext context =
FacesContextUtils.getRequiredWebApplicationContext (FacesContext.getCurrentInstance
( ));

ApplicationContext jpaCtx = (ApplicationContext)context.getBean("bean.factory");

//now there is an access to service beans
ICustServService srv = (ICustServService)jpaCtx.getBean("CustServService");

List<Customer> list = srv.findAllCustomers( );

```

To generate the Spring files and update the configuration:

1. To initiate the Spring artifact generation, right-click on the project name in the Project Explorer, and select **New > Generate Spring ORM Classes**. This opens the New Spring ORM Classes > Spring Services Definition wizard page.

Notice that the Spring bean configuration file field has been filled in when you added the Spring facet to your project. If you wish to use another file, click **Manage Spring bean configuration**, and then make your selection.

2. Click **Add** on the New Spring ORM Classes > Spring Services Definition dialog. This opens the New ORM Service dialog.
3. On the New ORM Service dialog, provide the following information:
 - Enter the service name.
 - Specify your project in **Persistence unit**. Note that you can select the persistence unit only when the service is created for the first time. Once created, you can add or remove entities from the existing persistence unit, but cannot switch to another persistence unit. To access another persistence unit, you create a new Spring Bean configuration and add a new service to it.
 - Select one or more entities from the ORM entities list.
 - Specify the exception that will be thrown by the service methods.
 - Provide the package information for the service and DAO classes and interfaces.
 - Specify the name for the service and DAO classes and interfaces (you may want to accept default values).
4. Click **OK** to continue.
5. Verify that the New Spring ORM Classes > Spring Services Definition dialog correctly displays the service name and the entities that you selected in the previous step, and then click **Next**. This opens the New Spring ORM Classes > Implementation Strategy wizard.
6. Select the implementation strategy for your Spring ORM service and DAO classes, and then click **Finish**.

The Spring classes are generated in the package that you specified. The bean classes contain methods for standard create, read, update, and delete operations.

Note that only the DAO implementation class is dependent on a specific framework. This lets you change ORM frameworks quickly, without rewriting client code.

11.2 Generating Web Services for Spring Service Beans

You can generate a WebLogic web service from an existing Spring service bean for your JAX-WS Web service project.

To generate the web service:

1. Add the WebLogic Web Services facet to an Eclipse project configured with the Spring 2.5 facet and containing Spring service beans.

For more information, see [Section 11.1, "Generating Spring Artifacts."](#)

2. In the Project Explorer, right-click your project, select **New > Other**, and then select **Oracle > WebLogic > WebLogic Web Service** for Spring Beans on the New dialog. This opens the Select Spring Service Bean dialog.
3. On the Select Spring Service Bean dialog, specify the Spring bean configuration. The list is populated with Spring service beans defined in the configuration file. Select a service bean and service methods from the service interface that you want to expose as WebLogic Web Service, and then click **Next**.

Note: While one service bean can implement multiple interfaces, you can only select methods from one interface at a time. To expose methods in other interfaces, you need to generate a separate Web service. Also, Spring ORM service beans implemented in Spring 1.*n*-style based on the `JpaTemplate` and `TransactionProxyFactoryBean` are not supported, therefore it is recommended that you upgrade your service bean to Spring 2.0-style using annotation-based transaction management.

4. On the Web Service Class Name page, you may choose to specify the Web service class name, package name and output directory, and then click **Finish**.

Upon completion, Eclipse creates the WebLogic Web Service class which is automatically wired up with Spring bean and delegate service calls to the Spring bean implementation.

Coherence Support

Coherence provides replicated and distributed (partitioned) data management and caching services on top of a peer-to-peer clustering protocol.

OEPE provides support for Oracle Coherence. For more information, see the Coherence information, which is available on Oracle Technology Network at <http://www.oracle.com/technetwork/middleware/coherence>.

This document contains the following sections:

- [Coherence Tooling: Configuring Projects for Coherence](#)
- [Working with Coherence \(GAR\) Applications](#)

12.1 Coherence Tooling: Configuring Projects for Coherence

This section describes how to work with Coherence facets.

12.1.1 Configuring Coherence Facet

Use the context menu and the properties dialog to add and configure your facets.

To configure your project for Coherence:

1. If you didn't add the Coherence facet when you create a project, you can add the facet to an existing project by right-clicking your project in the Project Explorer and selecting Properties from the drop-down menu. This opens the Properties dialog.
2. In the Properties dialog, select Project Facets on the left panel, and then select Oracle Coherence from the Project Facet list.

Click **Further configuration required** to open the Modify Faceted Project > Coherence dialog.

If the Coherence library is listed as a user library, proceed as follows:

- Select the Coherence library from the list.
- Accept default settings.
- Click **OK**.

If you are using Oracle WebLogic Server 12c (12.1.3) runtime, Coherence might already be installed on your machine if you chose this option when you installed. In this case, the Coherence Library in Fusion Middleware Install option is available in the **Library > Type** list.

If the Coherence library is not listed, you need to either point to an existing installation, or download the library.

To point to an already downloaded Oracle Coherence library:

- a. Click **Manage User Library** to open the Preferences > User Libraries dialog.
- b. Select a Java user library in the form of the `coherence.jar`, which represents the entire library.

Note that the Coherence JAR files reside in the `\coherence\lib` directory of the Oracle Coherence installation. Also note that the same dialog allows you to fine tune your Coherence library configuration.

- c. To obtain the Coherence library, click **Download Library**. This opens the Download Library dialog.
3. Click **OK** on the Modify Faceted Project > Coherence dialog.
4. Click **Apply > OK** to complete the configuration.

12.1.2 Editing Coherence Launch Configuration

The Coherence launch configuration editor sets the most commonly used Coherence command line arguments.

To modify the Coherence launch configuration for your project configured with the Coherence facet:

1. Right-click your project in the Project Explorer, and then select **Run As > Run Configurations** from the drop-down menu to open the Run Configurations dialog.
2. Select Coherence Launch Configuration from the list. The right pane of the Run Configurations dialog displays instructions on how to use this dialog to configure Coherence launch settings. You follow these instructions by right-clicking the configuration node on the left pane and making an appropriate choice.
3. When a new configuration is created and selected, you can edit settings from the right pane as follows:
 - On the Main tab, define your project and its main class. Note that the **Project** and **Main class** fields are populated with default values and cannot be left empty.
 - On the Coherence tab, define all the general, XML, and other system properties:
 - The General subtab lets you define properties related to topology, logging, and network configurations.
 - The XML subtab lets you set the management type (such as local or remote), as well as configure JMX RMI, HTTP, and the refresh options.
 - The Other subtab lets you set a variety of properties.

The Coherence launch configuration is populated with default settings from the `tangosol-coherence.xml` file in the `coherence.jar` file merged with the `tangosol-coherence-override.xml` file currently in the classpath, and, finally, with the VM arguments.

- Complete the remaining tabs of the Run Configurations dialog.
- Click **Apply > Run** on the Run Configurations dialog to complete the procedure. The Console view will display the Coherence run output.

The created Coherence XML files, such as `coherence-cache-config.xml`, `pof-config.xml`, and `tangosol-coherence-override.xml`, reside in your project's `src` directory.

12.1.3 Editing Coherence Operational Configuration

OEPE provides an editor that you can use to modify the generated `tangosol-coherence-override.xml` Oracle Coherence deployment descriptor.

To modify operational configuration elements defined in the `tangosol-coherence-override.xml` file:

1. In the Project Explorer, right-click `src/tangosol-coherence-override.xml`, and select **Open With > Coherence Override** to open the Coherence Override editor.
2. In the editor, select the following settings to modify:
 - Cluster
 - Logging
 - Configurable Cache Factory
 - Management
 - Security

For information on Oracle Coherence operational configuration deployment descriptors and their elements, see "Operational Configuration Elements" in *Developing Applications with Oracle Coherence*.

12.1.4 Editing Coherence Cache Configuration

OEPE provides an editor that you can use to modify the generated `coherence-cache-config.xml` Oracle Coherence deployment descriptor.

To modify cache configuration elements defined in the `coherence-cache-config.xml` file:

1. In the Project Explorer, right-click `coherence-cache-config.xml`, and select **Open With > Coherence Cache Configuration** to open the Coherence Cache Configuration editor.
2. In the Coherence Cache Configuration editor, you can modify the following settings:
 - General
 - Cache Scheme Mappings
 - Caching Schemes

For more information on Oracle Coherence cache configuration elements, see "Cache Configuration Elements" in *Oracle Coherence Developer's Guide*.

12.2 Working with Coherence (GAR) Applications

This section describes how to create Coherence applications, export them, deploy them, and locate deployed Coherence applications.

12.2.1 Creating Coherence Applications

With the release of WebLogic 12.1.3, Eclipse now supports the GAR (GridArchive) mechanism, a new packaging mechanism for Coherence cache configurations and classes.

You can create a new Coherence application using the Oracle Coherence Application wizard. The wizard creates a Coherence GAR project and gives you the option to add it to a J2EE EAR project.

To create a Coherence Application:

1. Choose **File > New > Other > Oracle > Coherence > Oracle Coherence Application**. The Oracle Coherence Application dialog appears.
 - a. Enter the project name. The project name is the name of the GAR project module in the application. The project name you enter here will be the name of the application. If you choose to add the project to the EAR, the EAR name will be generated based on the GAR project name.
 - b. Select your target runtime and server configuration details. The Coherence application requires that you use the Oracle WebLogic server 12.1.3.
 - c. Check to add the GAR project to the EAR. If you choose to add the GAR project module to EAR, a reference of the GAR module will be added to `weblogic-application.xml` in the EAR. Adding the project to the EAR allows the GAR module access to resources defined in the EAR, as well as other J2EE modules such as WAR and EJB.

Note: Only one GAR module can exist in an EAR project. Only Coherence configuration and classes can be included in GAR. No EJBs are allowed in GAR.

- d. Check to add the project to working sets. Working sets help you contextually organize resources across projects. Working sets are a subset of files/classes/folders/projects and represent development workflows.
2. Click **Next**. The Java Configure project for building a Java application dialog appears. Add or edit the source path.
 3. Click **Next**. The Coherence library configuration page appears. You have the option to change the Coherence library type to a user library, by specifying an Eclipse User Library containing the `coherence.jar`. Specifying the user library compiles the GAR project against the user library, instead of against the JARs from the WebLogic server, or Fusion Middleware installation. You can also select which default Coherence configuration files to create. When configuration files are created as part of the GAR, files such as `coherence-cache-config.xml` and `pof-config.xml` are placed in the root directory. For an overview of the Coherence default configuration files, see *Developing Applications with Oracle Coherence*.

12.2.2 Exporting a Coherence Application

You can export your Coherence project as either an EAR or a GAR file using the right-click context menu.

To export a Coherence Application:

1. Select your Coherence or EAR project node.
2. Right-click and select **Export**. The Export dialog appears.

3. Select an export destination and click **Next**.
4. Complete the destination details and click **Finish**.

12.2.3 Deploying a Coherence Application

You can deploy your Coherence application to a WebLogic server directly from OEPE, or you can export the application in an EAR or GAR and then deploy it to the WebLogic server with the Admin console, WLST, and weblogic.Deployer.

To deploy a Coherence Application:

1. Select your Coherence GAR or EAR project node.
2. Right-click and select **Run As > Run On Server**. The Run On Server dialog appears.
3. Select your server. Click **Next**.
4. Add or remove projects. Click **Finish**.
5. Once deployed successfully to the WebLogic server, the GAR application can be found under WLS server instance in the Servers View, under the Published Module node.

Note: Standalone GAR projects (not part of an EAR) require changing WLS Server publishing mode to the Exploded Archive Mode, in the server properties page.

12.2.4 Locating Your Deployed Coherence Application

To locate your deployed Coherence Application:

1. Select the WebLogic server instance in the servers view.
2. Expand the Published Modules, under the WebLogic server and locate the newly deployed Coherence application.

Or

Select the WebLogic server, right click and select **Goto > Admin Console** and locate the Coherence application.

Web Application Development Support

Web Application Development Tools is a set of plugins for the Eclipse IDE designed to help develop, deploy, and debug Web applications for Oracle WebLogic Server.

This document contains the following sections:

- [Using AppXray Technology](#)
- [Configuring JSF Projects](#)
- [Configuring JSTL Projects](#)
- [Configuring Projects for Apache Trinidad](#)
- [Configuring Projects with External Resources](#)
- [Creating a JSF Project From an Existing Web Project](#)
- [Using the Web Page Editor](#)
- [Editing Tags Using Property Sheets](#)
- [Using the Web Page Editor Palette](#)
- [Enabling Localization in the Web Page Editor](#)
- [Creating JSF HTML Tags](#)
- [Generating Struts Artifacts](#)
- [Supported Versions](#)

13.1 Using AppXray Technology

AppXray technology analyzes the JSP pages, Java source files, resource bundles, and Web configuration files of your application and uses this information to provide validation and consistency checking across many layers of the application, as well as source code completion and hyperlink navigation.

You can enable AppXray for your existing Web projects, or use it with your new dynamic Web projects for which it is enabled by default. Once available, AppXray will build its application database that allows it to track artifacts and populate it. If you have created a project using standard Web technologies, or if your project is JSP or JSF-enabled, those artifacts will be detected and added to the database. If you have enabled AppXray on an existing project, then artifacts in that project will be detected and added to the database. AppXray then uses this database for validation. As you work with your application, AppXray will automatically maintain the application database. If this degrades performance, you can selectively disable automatic maintenance and rebuild the application database as required.

Errors detected by AppXray are displayed in the Problems view, whereas dependencies are graphically displayed in AppXaminer.

AppXray collects information about the following:

- The project's `web.xml` file
- Faces configuration
- JSP pages
- JSP artifacts
- JSF configuration artifacts
- Resource bundles
- CSS
- Image files

13.1.1 Enabling and Disabling AppXray

To enable or disable AppXray:

1. Right-click your project in the Project Explorer and select **Properties** from the drop-down menu. This opens the Properties dialog.
2. In the Properties dialog, select the AppXray node. On the editor pane on the right, click **Configure workspace settings**, and then use the expanded **Preferences > AppXray** dialog to enable settings specific to your project, as well as fine-tune the technologies that you want AppXray to include in dependency discovery and collection, both of which are AppXray features used for finding missing resources, as well as collecting information about them (collection):
 - The role of discovery is to find variables in JSP pages. For example, disabling JSF discovery will cause managed beans to no longer appear in the Data Palette view.
 - Collection deals with all items that are available in the AppXaminer view (such as broken links, missing resources, JSP variables) and provides the navigation between files (for example, from a JSF `commandLink` tag to the section of the `faces-config.xml` file that defines the navigation rule).

You can disable collection and leave discovery enabled for a particular technology, but not vice versa.

Note: You cannot modify default project-level settings.

13.1.2 Visualizing AppXray Dependencies

With AppXray, you can expect to have a view of all Web project dependencies. By viewing these interrelationships graphically, missing resources and broken links become immediately apparent and application structure diagrams can be created by drilling down through the layers of your application.

To view dependencies for a Web project:

1. Right-click your Web project in the Project Explorer and choose **Show AppXray Dependencies** from the drop-down menu. This opens the AppXaminer view.

AppXaminer analyzes page, action, class, and many other dependency types. Initially all dependency types are active but can be easily reduced. AppXaminer

displays a graph of dependencies between technology-specific artifacts in the project. You can drill down into additional detail on particular references: each dependency link has a number that indicates how many references the focal node contains to the referenced file, with upstream dependency nodes having an indicator on the source end of the link, and downstream nodes having the indicator on the destination end of the link. By clicking the number, you bring up a detail showing the nature of each reference.

By clicking one of the references you navigate to the corresponding reference location in the referring file.

You can also right-click on a resource displayed in the AppXaminer diagram and select one of the following options from the drop-down menu:

- **Open** - to open the default editor on the file represented by the resource node.
 - **Show AppXray Dependencies** - to refocus the view on the selected node. Note that this option is disabled for the focal node.
 - **Expand All** and **Collapse All** - to expand or collapse nodes. Note that only one or the other is ever enabled, depending on the corresponding state of the node's expand/collapse button. If the node does not have an expand/collapse button, both options are disabled. Since the focal node potentially has two expand/collapse buttons, the options will affect both.
2. To display the chain of dependencies for the application, in AppXaminer view click the plus signs next to a dependency to drill down into the project and reveal the page structure of the application. Note that the Eclipse Outline view (which you open by selecting **Window > Show View > Outline** from the top-level menu) is synchronized with AppXaminer. If the AppXaminer diagram becomes too big, you can use the Outline view to scroll around the diagram.
 3. You can filter dependencies by clicking the **Filter Dependency** button located in the upper right-hand corner of AppXaminer to list the types of dependencies that can be tracked. This opens the Filter Dependency dialog which lets you filter either the project technology-specific artifacts, or artifacts by file extension.

You can define your custom filter extensions on the Filter Dependency > File Extensions dialog. By filtering out dependencies, you reduce the number of dependencies displayed in the diagram.

Notes: ■ The artifact types available for display and filtering depend on the technologies used in the project.

- All artifact types are shown by default.
 - All filter settings persist across invocations, for each technology type.
-

13.2 Configuring JSF Projects

OEPE provides a JSF project facet with the following functionality that you can use to configure your JSF projects:

- Support for Oracle WebLogic Server shared library
- Support for downloadable user library
- Library version validation
- JSP templates

- Notification that files will be deleted during uninstall of the facet

To add JSF capabilities to your Web project with Oracle Weblogic Server runtime support:

1. Right-click your Web project in Project Explorer, select **Properties** from the drop-down menu, and then select **Targeted Runtimes** from the list on the Properties dialog. Ensure that you are using Oracle WebLogic Server 11gR1 (10.3.2) or later as your target runtime.
2. On the Properties dialog, highlight **Project Facets** node and select **JavaServer Faces facet** from the list, specifying either version 1.2 or 1.1.
3. On the same Properties dialog, ensure that WebLogic Web App Extensions facet with one of 1.3.2, 1.3.1 or 1.3 versions is selected, and then click **Further configuration available**. This opens the **Modify Faceted Project > JSF Capabilities** dialog.
4. Complete this dialog by specifying JSF implementation library information, as well as configuration file name and servlet parameters.

When completing the **Modify Faceted Project > JSF Capabilities** dialog, you have the following two options with regards to which JSF implementation library to add to your project:

- a. **WebLogic Shared Library** - this option is available only when your project is configured with WebLogic Web App Extensions facet. If you click **Manage WebLogic Shared Libraries**, a **Preferences > Shared Libraries** dialog opens allowing you to modify library information, as well as to remove existing and add new libraries to your project.
- b. **User Library** - allows you to either download a specific version of the JSF library (you will be prompted to accept an appropriate license), or add the library from a local install (JAR file).

The system validates the selected library and will notify you if any of the following occurs:

- The required JSF class cannot be found in the library (for example, if you select a non-JSF user library).
- The selected library version and the facet version are not compatible (for example, if a JSF 1.1 library is selected for a JSF 1.2 facet).
- The library version cannot be read.

You can click **Download library** to initiate the download of the user library of your choice. Note that if your machine is located inside of a network, which requires a proxy to access outside resource such as the Internet, the download may fail. In this case, reconfigure your Eclipse IDE proxy settings using **Window > Preferences > General > Network Connections**, and try again.

If you select Include libraries with this application, the libraries will be copied to your project's `WEB-INF/lib` directory at publish time.

5. Click **OK** on the Modify Faceted Project > JSF Capabilities dialog to save your changes.

Upon completion, the following configurations of your project have taken place:

- `web.xml` has been configured depending on whether you chose Sun-RI or Apache MyFaces implementation.
- The following JSF templates are installed:

- `faces-config.xml` configuration file in your project's `WebContent/WEB-INF/config` directory.
 - `index.jsp` JSP file in your project's `WebContent` directory.
 - `application.properties` resource bundle in your project's `src/resources` directory.
- `faces-config.xml` file is configured as follows:

13.2.1 Supported JSF Libraries and Versions

OEPE provides support for JSF 1.2 and 1.1 versions.

Table 13–1 maps the supported JSF versions to three types of libraries:

Table 13–1 Supported JSF Versions

Supported JSF Version	Oracle WebLogic Server Shared Library	Downloadable Lib
JSF 1.2	Supported	Supported: MyFaces, Sun-RI
JSF 1.1	Supported	Supported: MyFaces, Sun-RI

13.2.2 Creating a Faces Configuration File

OEPE provides support for Faces configuration resource file by allowing you to create and edit a Faces Configuration File.

To create a new Faces configuration file for your JSF dynamic Web project:

1. Right-click the Project Explorer and choose **New > Other**. Alternatively, choose select **New > Other** from the top-level menu's File or from the AppXplorer view drop-down menu. This opens the New dialog.
2. On the New JSF Configuration File > JSF Configuration File Properties dialog specify the location for your new file (in the `WebContent/WEB-INF` folder) and provide a file name.
3. Click **Finish** to save your changes.

A new Faces configuration XML file containing the `faces-config` declaration is created with the given name and in the specified location.

The project's `web.xml` file is updated with a new path string in the value of the `javax.faces.CONFIG_FILES` context parameter. If the `javax.faces.CONFIG_FILES` context parameter does not exist, it is created.

Alternatively, in the Project Explorer you may right-click your JSF project's Faces Configuration directory and choose **New JSF Configuration File** from the drop-down menu.

13.2.3 Using the Faces Configuration Node

Using your JSF dynamic Web project's Faces Configuration node, you can create the following:

- Managed beans
- Navigation rules
- Converters
- Validators

13.2.3.1 Creating a New Managed Bean

You can create a managed bean in your Web application to bind the bean properties and business logic to the user interface components.

To create a managed bean:

1. In the Project Explorer, expand your JSF dynamic Web project's Faces Configuration node, right-click **Managed Beans**, and then select **New Managed Bean**. This opens the New Managed Bean Wizard dialog which allows you to create a managed bean from an existing or a newly created Java class.
2. If you choose to create a managed bean from an existing class:
 - a. On the New Managed Bean Wizard > Java Class Selection dialog click **Browse** to open the Select Type dialog. In this dialog, start typing the bean class name in the **Select entries** field, then select the class from the list and click **OK**.
 - b. Proceed by clicking **Next** on the New Managed Bean Wizard > Java Class Selection dialog to open the New Managed Bean Wizard > Managed Bean Configuration dialog.
 - c. Click **Next** to review the summary page, and then click **Finish**. Notice that your new bean has been added under the Faces Configuration > Managed Beans node.
3. If you choose to create a new Java class for your managed bean:
 - a. On the New Managed Bean Wizard > Java Class Selection dialog select **Create a new Java class**, and then click **Next** to open the New Managed Bean Wizard > Java Class dialog.
 - b. Complete this dialog, and then click **Next** to open the New Managed Bean Wizard > Managed Bean Configuration dialog. On this dialog, click **Next** to proceed to the summary page, and then click **Finish**. Notice that your new bean has been added under the Faces Configuration > Managed Beans node.

13.2.3.2 Creating a New Navigation Case

You create navigation rules to link Web pages. Each navigation rule can contain one or more navigation cases.

To define navigation rules and cases:

1. In Project Explorer, expand your JSF dynamic Web project's Faces Configuration node, right-click **Navigation Rules**, and then select **New Navigation Rule**. This opens the New Navigation Rule dialog which allows you to create a new navigation rule by providing an either new or existing JSP source page, and then defining destination pages.
2. If you choose to create your navigation rule using an existing JSP page as a source page:
 - a. On the New Navigation Rule dialog click **Browse for JSP** to open the Select JSP File dialog and select the page.
 - b. Define one or more navigation cases from the source page by clicking **New** on the New Navigation Rule dialog to open the New Navigation Case dialog.

Note that the destination page for your new navigation case can be an existing JSP page that you select using the Select JSP File dialog, or a new JSP page.
 - c. If you need to, create a new JSP. Click **Create a New JavaServer Page** on the New Navigation Case dialog to open the New JavaServer Page > JavaServer

Page dialog. Using this dialog, select a package and provide a name for your new JSP page, and then click **Next**.

- d. On the next New JavaServer Page > Select JSP Template dialog select a template to serve as initial content of your new JSP page, and then click **Finish**.

Click **OK** on the New Navigation Case dialog.

3. If you choose to create your navigation rule using a newly create JSP page as a source page, click Create a New JavaServer Page on the New Navigation Rule dialog to open the New JavaServer Page > JavaServer Page dialog.

Notice that your new navigation rule has been added under the Faces Configuration > Navigation Rules node.

13.2.3.3 Creating a New Converter

You can create a converter for your Web application. A converter is used in the conversion model, where each component can be associated with the server-side model object data. The component data can have two views: model and presentation. You can enable conversion of the data between the model view and the presentation view.

To create a converter:

- In the Project Explorer, expand your JSF dynamic Web project's Faces Configuration node, right-click Converters, and then select **New Converter**. This opens the New Converter dialog where you can define a new converter.

Notes: ■ The implementation class of your converter must be an extension of the `javax.faces.convert.Converter`.

- The implementation class of your converter must be an extension of the `javax.faces.convert.Converter`.
-

13.2.3.4 Creating a New Validator

To create a validator for your JSF project:

1. In the Project Explorer, expand your JSF dynamic Web project's Faces Configuration node.
2. Right-click Validators, and then select **New Validator**. This opens the New Validator dialog where you define the new validator.

Note: The implementation class of your validator must be an extension of the `javax.faces.validator.Validator`.

13.2.4 Using the Faces Configuration Editor

The Faces configuration editor allows you to modify the `faces-config.xml` file to define and edit page navigations, managed beans, components, converters, validators, render-kit, as well as to perform other element configurations.

You use the editor as follows:

In the Project Explorer, right-click your JSF dynamic Web project's Faces Configuration node, and select **Open**. This opens the Faces Configuration Editor' start page. This page enables access to useful resources such as documentation and help topics.

Use the following tabs to edit the Faces configuration:

- Overview tab that Figure 26 shows offers the compact presentation of all editable information. Double-clicking an element opens a relevant editor tab to enable editing.
- Overview tab that Figure 26 shows offers the compact presentation of all editable information. Double-clicking an element opens a relevant editor tab to enable editing.

You can create navigation rules by dragging and dropping JSP pages onto the canvas in one of the following ways:

- Directly from the Project Explorer.
- From the Palette by selecting **Nodes > Page** to open the Select JSP File dialog that Figure 28 shows.

To draw links between pages, use the Palette's Link tool.

You can reposition elements on the canvas. In addition, you can modify the elements' properties using the Quick Edit editor that displays properties of the selected element.

- Managed Bean tab allows you to modify configuration of managed beans in your project.
- Component tab enables editing of information about components, converters, validators, and render kits that you defined for your project.
- Others tab lets you edit various Faces configurations that are not covered in any of the other tabs.
- Source tabs provides access to the `faces-config.xml` file itself.

13.2.5 Understanding JSF Resource Bundles

If you would like your JSF application to use a resource bundle, you need to register it with the application. For JSF projects, when they are configured with the JSF facet, an `application.properties` file is automatically created in the `src/resources` folder. As one of the uses of a resource bundle is to enable localization, a `loadBundle` tag is typically added to each JSP page you want to localize:

```
<f:loadBundle basename="resources.application" var="bundle"/>
```

13.3 Configuring JSTL Projects

OEPE provides a JSTL project facet. You can add JSTL support to a new or existing Web project with Oracle Weblogic Server runtime support.

To add JSTL support:

1. Right-click your Web project in Project Explorer, select **Properties** from the drop-down menu, and then select **Targeted Runtimes** from the list on the Properties dialog. Ensure that you are using Oracle WebLogic Server 11gR1 (10.3.2) or later as your target runtime.
2. On the Properties dialog, highlight Project Facets node and select **JSTL facet** from the list, specifying either version 1.2 or 1.1.
3. On the same Properties dialog, ensure that WebLogic Web App Extensions facet with one of 1.3.2, 1.3.1 or 1.3 versions is selected, and then click **Further**

configuration available. This opens the Modify Faceted Project > JSTL Library dialog.

4. Complete this dialog by specifying JSTL library information.

When completing the Modify Faceted Project > JSTL Library dialog, you have the following two options with regards to what type of JSTL library to add to your project:

- **WebLogic Shared Library** - this option is available only when your project is configured with WebLogic Web App Extensions facet. If you click Manage WebLogic Shared Libraries, a Preferences > Shared Libraries dialog opens allowing you to modify library information, as well as remove existing and add new libraries to your project.
- **User Library** - allows you to either download a specific version of the JSTL library (you will be prompted to accept an appropriate license), or add the library from a local install (JAR file).

The system validates the selected library and will notify you if any of the following occurs:

- The required JSTL class cannot be found in the library (for example, if you select a non-JSTL user library).
- The selected library version and the facet version are not compatible (for example, if a JSTL 1.1 library is selected for a JSTL 1.2 facet).
- The library version cannot be read.

You can click **Download library** to initiate the download of the user library of your choice. Note that if your machine is located inside of a network, which requires a proxy to access outside resource such as the Internet, the download may fail. In this case, reconfigure your Eclipse IDE proxy settings using **Window > Preferences > General > Network Connections**, and try again.

5. Click **OK** on the Modify Faceted Project > JSTL Library dialog to save your changes.

Upon completion, your project's `web.xml` will be configured.

13.3.1 Supported JSTL Libraries and Versions

OEPE provides support for JSTL 1.2 and 1.1 versions.

The following table maps the supported JSTL versions to three types of libraries:

Table 13-2 Supported JSTL Versions

Supported JSTL Version	Oracle WebLogic Server Shared Library	Downloadable Library	User Library
JSTL 1.2	Supported	Supported	Supported
JSTL 1.1	Supported	Supported	Supported

13.4 Configuring Projects for Apache Trinidad

OEPE provides an Apache Trinidad project facet with the following functionality that you can use to configure your JSF projects:

- Support for downloadable user library
- Library version validation

- JSP templates
- Notification that files will be deleted during uninstall of the facet

To configure your Web project for Apache Trinidad:

1. Either add the Trinidad facet when you create the project, or add the facet to an existing project by right-clicking your project in the Project Explorer and selecting **Properties** from the drop-down menu. This will open the Properties dialog, as Figure 1 shows.
2. On the Properties dialog, select **Project Facets** on the left panel, and then select **Trinidad** from the Project Facets list. Note that the Trinidad 1.2 facet requires JavaServer Faces version 1.2.

To configure your project to use Trinidad 1.2 library, select the Trinidad 1.2 facet, and then click **Further configuration required**. This will open the Modify Faceted Project > Trinidad Library dialog, as Figure 2 shows. If Trinidad 1.2 is not listed, then click **Download Library** icon to indicate your intention to obtain the library. This will open the Download Library dialog, as Figure 3 shows.

3. Select Trinidad 1.2 library and the destination folder, and then click **Next**. Note that if you select a library other than Trinidad, or a mismatch between a Trinidad facet version and library version is detected, a notification message will be displayed alerting you of the error.
4. On the next screen, accept the term of the Apache license, and click **Finish** to trigger the download. Figure 4 shows Modify Faceted Project > Trinidad Library dialog after the Trinidad library has been downloaded.

Upon completion, the following configurations of your project have taken place:

- web.xml has been modified.
- index.jsp JSP file has been installed in your project's WebContent directory;
- application.properties resource bundle was installed in your project's src/resources directory.
- faces-config.xml file has been configured as follows:

```
<application>
<default-render-kit-id>org.apache.myfaces.trinidad.core</default-render-kit-id>
</application>
```

13.4.1 Trinidad Library Support by the Trinidad Facet

OEPE provides support for Trinidad 1.2 and 1.0 versions.

The following table associates the supported Trinidad versions with JSF versions:

Table 13–3 Supported Trinidad Versions

Supported Trinidad Version	Associated JSF Version
1.2	1.2
1.0	1.1

13.5 Configuring Projects with External Resources

If you have a project that uses external resources, for example Java classes and web resources such as JSP files, which are present outside of your workspace, or from other projects, you can configure the project so that WebLogic looks up these resources at runtime.

The features that you can use are:

- Using a wrapper dynamic Web project
- Using linked resources
- Configuring a deployment assembly

13.5.1 Using a Dynamic Project

If your project is not an Eclipse WTP project you need wrap it with a dynamic web project in order to deploy it to Oracle WebLogic Server. Once you have created the skeleton web project, you can use the linked resources technique (below) to link web resources to the corresponding Web content or Java source folder in the dynamic Web project. This project may contain a number of optional facets, one of which is the Web Service facet. Each dynamic Web project ultimately produces a J2EE module, and each J2EE module is included in the complete application's EAR file when the application is built for deployment. The contents of Web projects are accessed through URLs.

For more information, see "Creating a dynamic Web project" which is available in the Web Tools Platform User Guide in the Eclipse documentation.

13.5.2 Using Linked Resources

Linked resources are files and folders stored in a different location but which can be linked to your project. You can link both Web content such as JSP files and HTML files, folders or Java source folders into a web project using this method. For more information, see Linked resources which is available in the Workbench User Guide in the Eclipse documentation. For information about using linked resources, see Creating linked resources, in the Workbench User Guide in the Eclipse documentation.

13.5.3 Configuring a Deployment Assembly

The types of external resource you can configure using deployment assembly are:

- Archives available via a path variable, or on the local file system, or in the current workspace.
- Folders in the current workspace.
- Java build path entries.
- Projects in the current workspace.

To configure a project to use external resources:

1. Right-click your Web project in Project Explorer, select **Properties** from the drop-down menu, and then select **Deployment Assembly** from the list on the Properties dialog.

Figure 1 shows a web content folder from linked resources and an external JAR have been added to the deployment assembly.

2. Click **Add** to open the New Assembly Directive dialog, where you specify the external resources to be configured with your project.

3. Choose the type of external resource you want to add on this page and click **Next**.
4. On the next page, navigate to the specific external resource you want and click **Finish**.

13.6 Creating a JSF Project From an Existing Web Project

Using OEPE, you can create a new JSF project from an existing Web project if the existing project meets the following criteria:

- The source project is a non-Eclipse project and does not contain any Eclipse-specific files or folders.
- The source project's WebContent folder is not the project root.
- Somewhere in its directory hierarchy the source project contains a directory with a subdirectory called `WEB-INF`, which in turn contains a file called `web.xml`.

To create your project:

1. Right-click the Project Explorer and select **New > Project** from the drop-down menu. This opens the New Project dialog.
2. Select **Web > Dynamic Web Project from Existing Source** from the list, as Figure 1 shows, and then click **Next**. This opens the New Web Project dialog.
3. On the New Web Project > Project Location dialog that Figure 2 shows, specify your existing project's location, provide a name for your new project, select the compiler compliance level, select Oracle WebLogic Server 11gR1 (10.3.2) or later as your target runtime, and then click **Next** to continue.
4. On the next New Web Project > Java Settings screen that Figure 3 shows, either accept the default Java settings for your project or customize them, and then click **Finish**.

Upon completion, the new project from the existing source is created and is configured with the following facets:

- JSF 1.2
- JSTL 1.2
- jst.web 2.5 (depends on the version set in `web.xml`)
- jst.java 5.0 (depends on the compiler compliance level selected)
- wls.web 10.3 (depends on the server selected)

13.7 Using the Web Page Editor

You use the Web Page Editor to edit JSP and HTML files. It is a multi-page editor that provides the following:

- Design page that supports visual development.
- Source page that lets you edit text.
- Preview page

Using its toolbar, you can configure the editor to display both a Design and a Source page of the current document in either horizontal or vertical split modes, as Figure 1 shows. You can also configure it to display only either the Design or the Source page. Note that the currently-selected element is synchronized between all pages and views.

The following views are associated with the Web Page Editor:

- Properties view that enables editing of the most common attributes of tags using choice dialogs.
- Palette view that allows you to edit and create tags, such as HTML, JSP, and JSF.
- Outline view

The Web Page Editor also includes the Preview tab and a toolbar. The toolbar enables the following:

- Display of horizontally-split Design and Source pages
- Display of vertically-split Design and Source pages
- Display of Design page only
- Display of Source page only
- Underlining selected text
- Bolding selected text
- Italicizing selected text
- Making selected text appear small
- Making selected text appear large

To access the Editor, from your Web project open in the Project Explorer, right-click a page which you are planning to edit, and then select Open With > Web Page Editor from the drop-down menu, as Figure 2 shows.

13.7.1 Using the Design View

The Design view is editable and allows you to select elements and move them around on the page, display an element's properties in the properties editor, drop tags from the Palette onto the page, edit elements using the context menu, and so on.

Figure 3 shows a JSP page displayed in the design view with an input text selected and its properties displayed in the properties editor.

Note that some elements enable tag-specific editing through the context menu.

The Design view provides tooltips with the information on which element will be selected upon a mouse click.

In addition, the Design view allows you to visually inspect and select nonvisual child components, such as converters and validators. To do so, float the cursor over an element, and you will see these nonvisual child components as semitransparent icons at the top-right of the element.

Selecting the element and then clicking the "pin" icon that appears at the top-right of the element will then let you select the nonvisual child component by clicking the component's icon.

13.7.2 Using the Preview Tab

The Preview tab represents a non-editable view that closely emulates the Web page as it will be rendered at run time.

Figure 6 shows a preview of a JSP page.

13.7.3 Using the Source View

The Source view is editable and allows you to select elements, display an element's properties in the properties editor, drop tags from the Palette onto the page, and so on. Figure 7 shows a JSP page displayed in the Source view with an output text selected and its properties displayed in the properties editor.

13.7.3.1 Using the Content Assist

You can use the content assist feature of the Source view that lets you select tags from a list of available tags with descriptions. You activate the content assist by pressing **Ctrl+Space** key combination.

13.7.3.2 Using HyperLink

You can use the HyperLink feature of the Source view to open a managed bean's code. To do so, press **Ctrl+Click** key combination on the value attribute of a tag.

13.7.3.3 Using HoverHelp

The Source view offers the HoverHelp that displays information about the page elements, or on how to use them.

You can zoom in to the help topic by pressing F2.

13.7.4 Using the Outline View

The Outline view (see Figure 6) allows you to do the following:

- Visually inspect the document structure
- Change the selection of the current element
- Drag and drop elements within the view
- Modify elements and attributes using the context menu

13.8 Editing Tags Using Property Sheets

You use property sheets to quickly edit the most common attributes of various tags with assistance of choice dialogs. Property sheets offer collapsible sections and provide buttons that enable choice of values and binding to dynamic values.

The Property sheet (view) displays attributes and their values for a tag selected in the Design or Source page. The Property view consists of two tabs:

- All tab shows all attributes of a tag.
- General tab shows the most commonly used attributes of a tag, assuming this tag is one of JSF, JSP, JSTL core, JSTL formatting or HTML. For each attribute shown in the General tab, OEPE provides custom dialog editors that guide you in setting the values of the attribute by doing the following:
 - Choosing binding
 - Choosing a method
 - Selecting a navigation case
 - Selecting a file
 - Choosing a style class
 - Defining CSS style

- Choosing a resource bundle
- Choosing a validator
- Choosing a converter

The General tab also provides hyperlink on select attributes that help you navigate to the target resource.

You open a Property sheet (see Figure 1) by selecting Window > Show View > Properties from the top-level menu. By selecting a tag in the Source or Design view and then clicking on buttons located to the right of attribute edit fields on the Property sheet displays editor dialogs.

13.8.1 Choosing Binding

You use the Choose Binding dialog to create bindings to dynamic values. Figure 2 and Figure 3 show the Variables and Resources tabs of this dialog.

The Resources tab allows you to do the following:

- Select the resource bundle, if more than one exists.
- Select the resource key.
- Select whether the dialog should display the resource key or value. You use the View combo box to make this choice.
- Add a resource key by clicking the New Resource Key.

13.8.2 Choosing a Method

You use the Choose Method dialog that Figure 4 shows to select an existing or create a new method in the selected managed bean.

To create a new method, click New Validator Method, and then complete the New Method dialog.

13.8.3 Selecting a Navigation Case

You use the Select Navigation Case dialog that Figure 6 shows to specify a navigation case outcome. If there are no existing navigation cases listed, you may click New Navigation Case to open a dialog that will let you define a new navigation case (action) and either select an existing JSP page in the project, or create a new destination JSP page for your new navigation case.

13.8.4 Selecting a File

You use the File Selection dialog to select the source of a file, such as an image, CSS, or a JSP file. To open the dialog, click Select File.

13.8.5 Selecting a Style Class

You use the Choose Style Class dialog to select a style class for the selected tag.

13.8.6 Defining CSS Style

You use the CSS Style Definition dialog to define a CSS to apply to the tag. Using this dialog, you can define all elements of a style sheet.

Note that for the style to be applied, you need to register it with the application.

13.8.7 Choosing a Resource Bundle

You use the Resource Bundle Selection dialog to choose a resource bundle for your application (as defined by the `basename` attribute). The dialog displays a list of available resource bundles including the following:

- Resource bundles that reside in your project's `src` directory.
- Resource bundles in JAR files in your project's `WEB-INF/lib` directory.
- Resources bundles in JAR files in Shared Libraries.
- Resources bundles in JAR files in User Libraries.

Note that this dialog allows you to set the resource bundle defined by the `basename` attribute of the JSF Core `loadBundle` and JSTL Formatting `setBundle` tags.

Also note that if the `basename` attribute contains a valid value, it will be selected in the dialog by default.

13.8.8 Choosing a Validator

You use the Validator Id Selection dialog to choose a validator from a list of available validators, which includes the standard JSF validators and any validators that are defined in the `faces-config.xml` file. Note that through this dialog you set the value of the `validatorId` attribute of the JSF Core validator tag.

If the `validatorId` attribute already contains a valid value, this value is displayed separately at the top of the dialog.

13.8.9 Choosing a Converter

You use the Converter Id Selection dialog to choose a converter from a list of available converters, which includes the standard JSF converters and any converters that are defined in the `faces-config.xml` file. Note that through this dialog you set the value of the following attributes:

- `converterId` attribute of the JSF Core converter tag.
- `converter` attribute of JSF HTML `inputText` and `outputText` tags.

If the `converterId` or `converterId` attribute already contains a valid value, this value is displayed separately at the top of the dialog.

13.9 Using the Web Page Editor Palette

You use the Web Page Editor Palette to edit and create a variety of tags, such as HTML, JSP, JSF, and so on.

In addition to standard HTML and JSP tags, the Palette displays an item for each tag in the JSP tag libraries that are on the application's classpath. You drag and drop tags on to the Design or Source view to design Web pages. You can pin the Palette to be open, or set to automatically expand when the cursor is placed over it while it is in its collapsed state.

To access the Palette, from your Web project open in the Project Explorer, right-click a page on which you are planning to drop or edit tags, and then select **Open With > Web Page Editor** from the drop-down menu.

With the page open in the editor view, click Show Palette (a gray triangular button located at the top right corner of the editor) to display the Palette.

13.9.1 Displaying the Palette in External View

You can display the Palette outside of the Web Page Editor. To do so, right-click the Palette and select **Options > Show as External Palette** from the drop-down menu

To reverse the external view display and have the Palette displayed as a part of the Web Page Editor, click the **Close** button.

Notice that the external view allows you to filter the tag library nodes to define which ones the Palette should display. To do so, enter the filter text in the filter field.

13.9.2 Editing Tag Library Entries in the Palette

To add a tag, you select it from the library and then drag it from the Palette and drop onto the JSP page that is open in either the Design or Source view in the Web Page Editor. You can enter values for the tag attributes using the corresponding dialog that pops up upon the drop.

Alternatively, you can use a property sheet to add and edit tags.

13.9.3 Using the Data Palette

The Data Palette is a part of the Web Page Editor Palette and is always the last list item in the tag library list.

The Data Palette displays variables in scope for the page, as follows:

- Variables are displayed as children of a group and grouped by Page Variables and External Variables. If a group does not contain any member variables, the group is not displayed. The scope of a variable is indicated by an icon (green square for the application scope, blue triangle for the session scope, orange rectangle for the request scope, and blue circle for the page scope), as well as by the tool tip text that also provides information about the variable type, name, and data type.
- Fields are displayed as children of a variable and include fields with accessor methods indicated by a green circle and a tool tip text that lists the field name and data type.
- Fields of fields are displayed as children of the fields.

Note: The Data Palette automatically refreshes itself when variables are discovered. Also note that when the Palette is displayed as an external view and you have the ability to use filters, you can filter by "used variable only".

The Data Palette allows you to do the following:

- Add a variable to a page by right-clicking a variable and selecting **Insert in the page** from the drop-down menu.

This opens the Insert Fields dialog that lets you select a generator for content creation choose fields that you would like to generate and define the order in which they should appear.

- Navigate to a variable declaration by right-clicking the variable and selecting **Open Declaration** from the drop-down menu.

This opens the editor with the position of the variable declaration highlighted.

- Navigate to the value reference type declaration by right-clicking the variable and selecting **Open Type** from the drop-down menu.

This opens the appropriate Java class in the source editor or, if the source is not available, in the read-only Class File Editor.

Notes: ■ If the type is an array type, the component type is used. For example, if it is `java.lang.String[]`, you would navigate to `java.lang.String`.

- If the type is a collection type, the container type is used.
 - The navigation is not available for primitive types.
-
-

- Create new external (artificial) variables by right-clicking a variable group node and selecting **New Artificial Variable** from the drop-down menu. This opens the New Artificial Variable dialog. You create a variable by entering its name and providing its data type.

13.9.4 Customizing the Palette

To customize the way the Palette displays tags, right-click a specific tag library node and select **Options > Customize**.

This opens the Customize Palette dialog.

Using this dialog, you can modify names and descriptions of tag library nodes, specify whether or not a particular node should be displayed, as well as import or export tag libraries.

You can also modify the Palette settings by right-clicking a specific tag library node and selecting **Options > Settings** to open the Palette Settings dialog.

Using this dialog, you can specify the font, layout, and icon size for a tag library node, as well as define drawer options.

13.9.5 Docking and Undocking the Palette

When the Palette is displayed as an external view, you may choose to change the display to the docked view in the Web Page Editor. To do so, click on the Palette tab and select **Fast View** from the drop-down menu.

Notice the Palette icon at the bottom left corner of the main IDE window.

13.9.6 Modifying the Display of the Palette

In addition to performing the Palette customization using the Customize Palette and Palette Settings dialogs, you can modify the way the Palette and its tag library nodes are displayed within the Web Page Editor as follows:

- Move it to the left or right side of the Web Page Editor by clicking the top area and dragging the cursor to the left or right.

Alternatively, you can move the Palette by right-clicking the top area and selecting **Dock On > Left** or **Dock On > Right** from the drop-down menu.

- Change the width of the Palette by right-clicking the top area and selecting **Resize** from the drop-down menu.

- Use the marquee tool. To do so, click the Marquee button located on the Palette toolbar.
- Apply the following changes to a tag library node by making selections from the drop-down menu with the cursor positioned on a particular node:
 - Expand the size of a tag library node to the size of the Palette by selecting **Maximize**.
 - Remove a tag library node from the Palette view by selecting **Hide**.
 - Undo the preceding changes by selecting **Restore**.
- Change the Palette layout or the size of icons for a tag library node by selecting appropriate options from the drop-down menu.

13.10 Enabling Localization in the Web Page Editor

To localize your Web application:

1. Register the application's resource bundle. For localization to take effect, you define multiple properties files (one for each language), such as `application_fr.properties` or `application_it.properties`, for example, in your project's `srs/resources` directory.
2. Localize the JSP page by binding the attributes of tags that display text on the page, as follows:
 - Select a tag in the JSP file and open a property sheet to display the tag's attributes in editable mode.
 - Click Bind to a dynamic value button located to the right of the field that you are editing to open the Choose Binding dialog, and then open the Resources tab.

The Resources tab displays either the available resource keys or their corresponding values (you set the display mode by making selection in the View box).
 - Using this dialog, you can create a new resource key in the bundle by clicking New Resource Key button a dialog that allows you to define a key and value combination to be added to the resource bundle.

For example, if the entered key is `welcome.back` with the value "Please, come back", the newly created resource would be displayed in the Choose Binding dialog.
3. You can visualize the resources in different locales in the Web Page Editor. To set the design-time locale for your project, use the Resource Locale dialog that you open by selecting a text in the Web Page Editor, and then selecting Text > Resource Locale from the top-level menu.

Once you choose a new locale and define a new resource key, the new key will be saved in the locale-specific properties file.

13.11 Creating JSF HTML Tags

OEPE provides tag drop support for JSF Web projects. This OEPE feature is made available through a series of wizard-style dialogs that are displayed during "specialty" drops that require not just simple tag editors, but tag-addition guides that assist you by offering appropriate attribute editors and default values. Note that you cannot use these editors to modify tags, but only to add `panelGrid`, `dataTable`, and `form` tags.

To access this OEPE functionality, open your JSF Web project in Project Explorer, right-click a JSP page on which you are planning to drop tags, and select **Open With > Web Page Editor** from the drop-down menu.

With the page open in the editor view, click Show Palette (a gray triangular button located at the top right corner of the editor) to display the Palette.

To add tags, you drag one of Panel Grid, Data Table, or Form components from the Palette to the body of your JSP page in the editor and complete New dialogs.

Note that you have an option of binding dialogs' fields to beans or bean properties by clicking buttons located to the right of most fields.

13.11.1 Adding a PanelGrid Tag

You add a panelgrid tag using the New Panel Grid dialog that provides tag attribute editors.

Using this dialog, in the Generation options field, specify whether you are adding a new panelGrid tag, or generating one and its content from data. If you choose the former, proceed with the following:

- In the Columns field, enter a number of columns.
- In the Width field, specify the width as an integer.
- In the Table Class field, specify the table style class.

If you choose the latter, the New Panel Grid dialog will change and you proceed as follows:

1. Specify the bean property by clicking **Select a Variable** button located to the right of the Bean property field to open the Choose Bean/Bean Property dialog.
2. Click **OK** on the Choose Bean/Bean Property dialog, and then click **Next** on the New Panel Grid dialog to open the New Panel Grid > Choose Fields dialog.
3. On the **New Panel Grid > Choose Fields** dialog, you may select the fields that you want to generate for your panel grid, and then click **Next**.
4. On the next **New Panel Grid > Choose Field Components, Labels and Ordering** dialog, you may select components to represent fields and the labels for these fields. In addition, you may define the order in which the fields will appear by clicking Up and Down buttons located to the right of the Field Selection area.
5. You may click **Next** to open the **New Panel Grid > Choose Options** dialog. On this dialog, define if your panel grid should have a header and a footer, and whether or not validation messages should be displayed for input fields, and if so, choose their style by clicking a button to the right of Style class field to open the Choose CSS Style Class dialog.
6. Upon completion, in the body element of your JSP page an empty `h:panelGrid` tag with the specified parameters is added.

13.11.2 Adding a dataTable Tag

You add a dataTable tag using the New Data Table dialog.

Using this dialog, in the Generation options field specify whether you are adding a new dataTable tag, or generating one and its content from data. If you choose the former, proceed with the following:

- In the Data Table area, provide the table ID, as well as to which UI component this table is to be bound by clicking the Bind to a dynamic value button located to the right of the Binding field and making selection of either variables or resources on the Choose Binding dialog.
- In the Data Table area, provide the table ID, as well as to which UI component this table is to be bound by clicking the Bind to a dynamic value button located to the right of the Binding field and making selection of either variables or resources on the Choose Binding dialog.
- In the Table Properties area, you may set the table's width and border size as integers, as well as specify classes for the table, row and column.

If you are generating a dataTable and its content from data, the New Data Table dialog will change and you proceed as follows:

- Specify the enumerable bean property by clicking Select an Enumeration button located to the right of the Enumerable field to open the Choose Enumerable Bean Property dialog.
- Enter the iteration variable name and type by completing appropriate dialogs. You may also choose the enumerable fields to generate and define their order, as well as insert a header and footer for your table. For information on how to do this, see [Section 13.11.1, "Adding a PanelGrid Tag."](#)

Upon completion, a `h:dataTable` tag with the specified parameters is added within the body element of your JSP page.

13.11.3 Adding a form Tag

You add a form tag using the New Form dialog.

Using this dialog, in the Generation options field specify whether you are adding a new form tag, or generating one and its content from data. If you choose the former, proceed with the following:

- In the Id field, enter the form ID.
- In the Binding field, specify to which UI component this form is to be bound by either entering the value, or clicking the Bind to a dynamic value button located to the right of the Binding field and making selection of either variables (of type `javax.faces.component.UIComponent`) or resources on the Choose Binding dialog.

If you choose to generate a dataTable and its content from data, the New Form dialog will change, and you proceed as follows:

- Specify an existing JSF managed bean by clicking the Select a Variable button located to the right of the Form bean field to open the Choose Bean/Bean Property dialog.

Click OK on the Choose Bean/Bean Property dialog.

- Optionally, you can specify a form action, such as a navigation case outcome or a method binding, by clicking button to the right of the Form action field. Clicking Select a value will open the Select Navigation Case dialog. If there are no existing navigation cases listed, you may click New Navigation Case to open the dialog that will let you to define a new action (navigation case).

To specify a method binding, click Bind to a dynamic value button located to the right of the Form action to open the Choose Method dialog that will let you select a method, if available, or add a new method in the selected bean.

You can either click Finish on the New Form dialog, or click Next sequentially to provide the following optional configurations:

- Fields - from the list of fields with accessor methods, select fields to generate.

Notes: ■Fields of fields are displayed recursively as children.

- Non-complex" first-level fields are selected by default; other fields, such as enumerated types and dates, are not selected because they are considered "complex".
-
-

- Field Components, Labels and Ordering - select components to represent fields and the labels for these fields. In addition, you may define the order in which the fields will appear by clicking Up and Down buttons located to the right of the Field Selection area.
- Options - define if your form should have a header and a footer, and whether or not validation messages should be displayed for input fields, and if so, choose their style by clicking a button to the right of Style class field to open the **Choose CSS Style Class** dialog.

13.12 Generating Struts Artifacts

You can use OEPE to automatically generate various Struts artifacts for your dynamic Web project. To do so, perform the following steps:

- Make sure you are using supported versions of Oracle WebLogic Server and Struts.
- Configure your project for Struts.
- Generate Struts files and update the configuration.

13.12.1 Configuring a Project for Struts

To configure your dynamic Web project for Struts:

1. Either add the Struts facet when you create the project, or add the facet to an existing project by right-clicking your project in the Project Explorer and selecting **Properties** from the drop-down menu. This will open the Properties dialog.
2. In the Properties dialog, select Project Facets on the left panel, and then select **Struts** from the Project Facet list. Note that the Struts facet requires dynamic Web module version 2.3 or later.

To configure your project to use Struts 1.3 library, select the Struts 1.3 facet, and then click Further configuration required... link. This will open the **Modify Faceted Project > Struts** dialog. If Struts 1.3 is not listed, then click Download Library icon to indicate your intention to obtain the library. This will open the Download Library dialog.

Note that you can either add the library from the local installation, or you can select a shared library provided by Oracle WebLogic Server (if this option is supported).

Select Struts 1.3 library and the destination folder, and then click **Next**. Note that if you select a library other than Struts, or a mismatch between a Struts facet version and library version is detected, a notification message will be displayed alerting you of the error.

On the next screen, accept the term of the Apache license, and click **Finish** to trigger the download.

3. Click **OK** on the **Modify Faceted Project > Struts Facet** dialog.
4. To complete the configuration of the Struts library for your project, click **Apply > OK** on the Properties dialog.

13.12.2 Generating Struts Files and Updating the Configuration

When added to your dynamic Web project, the Struts facet automatically generates the following artifacts and preforms the following configurations:

Configures the `web.xml` file, as follows:

- Adds an action servlet in a form of the `org.apache.struts.action.ActionServlet`.
- Adds a servlet-mapping (using the URL `*.do` pattern).
- Adds a config parameter for `struts-config.xml`.
- Adds Struts-specific configuration files, such as `struts-config.xml` and others, in your project's `WEB-INF/config` directory.
- Adds JSP template files and resource bundles containing the sample Struts code that you can deploy out of the box.

13.13 Supported Versions

OEPE provides support for various Oracle WebLogic Server versions, as well as the following Apache Struts versions:

- 1.3
- 1.2
- 1.1

The following table maps supported versions of Struts libraries and Oracle WebLogic Server shared libraries:

Table 13–4 Supported Versions of Struts Libraries

Struts Library	Oracle WebLogic Server Shared Library	Downloadable Library	User Library
Struts 1.3	Not supported	Supported	Supported
Struts 1.2	Oracle WebLogic Server versions 10.0 and 9.2 are supported	Supported	Supported
Struts 1.1	Oracle WebLogic Server versions 10.0 and 9.2 are supported	Supported	Supported

