

Oracle® SOA Suite

Developer's Guide

10g (10.1.3.1.0)

B28764-01

September 2006

Oracle SOA Suite Developer's Guide, 10g (10.1.3.1.0)

B28764-01

Copyright © 2006, Oracle. All rights reserved.

Primary Author: Ken Chu, Orlando Cordero, Mario Korf, Catherine Pickersgill, Robin Whitmore

Contributing Author: William Bathurst, Roman Dobrik, Raghu Kodali, Robin Zimmermann

The Programs (which include both the software and documentation) contain proprietary information; they are provided under a license agreement containing restrictions on use and disclosure and are also protected by copyright, patent, and other intellectual and industrial property laws. Reverse engineering, disassembly, or decompilation of the Programs, except to the extent required to obtain interoperability with other independently created software or as specified by law, is prohibited.

The information contained in this document is subject to change without notice. If you find any problems in the documentation, please report them to us in writing. This document is not warranted to be error-free. Except as may be expressly permitted in your license agreement for these Programs, no part of these Programs may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose.

If the Programs are delivered to the United States Government or anyone licensing or using the Programs on behalf of the United States Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the Programs, including documentation and technical data, shall be subject to the licensing restrictions set forth in the applicable Oracle license agreement, and, to the extent applicable, the additional rights set forth in FAR 52.227-19, Commercial Computer Software--Restricted Rights (June 1987). Oracle USA, Inc., 500 Oracle Parkway, Redwood City, CA 94065.

The Programs are not intended for use in any nuclear, aviation, mass transit, medical, or other inherently dangerous applications. It shall be the licensee's responsibility to take all appropriate fail-safe, backup, redundancy and other measures to ensure the safe use of such applications if the Programs are used for such purposes, and we disclaim liability for any damages caused by such use of the Programs.

Oracle, JD Edwards, PeopleSoft, and Siebel are registered trademarks of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

The Programs may provide links to Web sites and access to content, products, and services from third parties. Oracle is not responsible for the availability of, or any content provided on, third-party Web sites. You bear all risks associated with the use of such content. If you choose to purchase any products or services from a third party, the relationship is directly between you and the third party. Oracle is not responsible for: (a) the quality of third-party products or services; or (b) fulfilling any of the terms of the agreement with the third party, including delivery of products or services and warranty obligations related to purchased products or services. Oracle is not responsible for any loss or damage of any sort that you may incur from dealing with any third party.

Contents

Preface	xvii
Audience	xvii
Documentation Accessibility	xvii
Related Documents	xviii
Conventions	xix

Part I Getting Started With Oracle SOA Suite

1 Introduction to SOA and the Oracle SOA Suite

1.1	Overview of SOA and the Oracle SOA Suite	1-1
1.2	SOA Benefits	1-2
1.3	Using Oracle SOA Suite to Adopt SOA	1-3
1.3.1	Creating Services	1-4
1.3.2	Using an Enterprise Service Bus	1-5
1.3.3	Composing and Orchestrating Services	1-6
1.3.4	Automating Business Policies with Business Rules	1-7
1.3.5	Securing and Managing Services Using Oracle Web Services Manager	1-8
1.3.6	Deploying SOA Applications	1-9
1.3.7	Benefits of Oracle SOA Suite	1-10

2 The SOA Order Booking Application

2.1	Introduction to the SOA Order Booking Application	2-1
2.2	Installing the SOA Order Booking Demonstration Application	2-2
2.2.1	System Requirements	2-2
2.2.2	Application Installation Prerequisites	2-2
2.2.3	Installing the Database Schema	2-3
2.2.3.1	The SOA Order Booking Schema	2-3
2.2.4	Configuring the Container	2-5
2.2.5	Noting the Important Port Numbers	2-8
2.2.6	Creating Connections in Oracle JDeveloper	2-9
2.2.7	Deploying the Separate SOA Order Booking Applications	2-10
2.2.8	Configuring the ESB Port	2-16
2.3	An Overview of the SOA Order Booking Application	2-17

2.4	Working with the SOA Order Booking Application Code in Oracle JDeveloper	2-19
2.4.1	The SOADemo Application	2-19
2.4.1.1	The CreditService Project	2-19
2.4.1.2	The CustomerService Project	2-20
2.4.1.3	The FulfillmentESB Project	2-21
2.4.1.4	The OrderBookingESB Project	2-21
2.4.1.5	The RapidService Project	2-22
2.4.1.6	The SelectManufacturer Project	2-23
2.4.1.7	The SOAOrderBooking Project	2-23
2.4.2	The SOADemo-Client Application	2-24
2.4.2.1	The Assembly Project	2-24
2.4.2.2	The ClientModel Project	2-24
2.4.2.3	The CustomerService Project	2-25
2.4.2.4	The OrderService Project	2-26
2.4.2.5	The UserInterface Project	2-26
2.5	Taking a Closer Look at the SOA Order Booking Application	2-28
2.5.1	Registration Process	2-28
2.5.2	The Login Process	2-30
2.5.3	The Ordering Process	2-30
2.5.3.1	The Web Client	2-30
2.5.3.2	The OrderBookingESB Flow	2-32
2.5.3.3	SOAOrderBooking BPEL Flow	2-32
2.5.3.4	OrderFulfillmentESB Flow	2-46
2.5.4	Adding Security to the System	2-47
2.5.5	Monitoring the System	2-48
2.5.5.1	Oracle ESB Control	2-48
2.5.5.2	Oracle BPEL Control	2-49
2.5.5.3	Oracle Enterprise Manager Control Console	2-51
2.5.5.4	Oracle Web Services Manager Monitor	2-51

Part II Developing SOA Systems

3 Designing an SOA Application

3.1	Introduction to Designing an SOA Application	3-1
3.2	Defining the Project in the Project Scope	3-3
3.2.1	Creating a Service Portfolio Plan	3-4
3.2.2	Determining Business and System Objectives	3-7
3.2.3	Determining Business and Operational Requirements	3-8
3.2.4	Conducting a Discovery Phase	3-12
3.2.5	Defining the Project	3-15
3.2.6	Creating a Solution Specification	3-20
3.3	Designing the Project in the Application Scope	3-21
3.3.1	Designing a Process Flow	3-21
3.3.2	Designing Services	3-24
3.3.3	Creating a Physical Design	3-26

4 Building and Using Application Services

4.1	Introduction to Business Services.....	4-1
4.2	Creating JPA Entities that Map to Database Tables.....	4-2
4.2.1	What Happens When You Create JPA Entities From Database Tables.....	4-2
4.2.2	What You May Need to Know About Creating Entities From Tables.....	4-5
4.3	Configuring Persistence Units	4-5
4.3.1	How to Create a Persistence Unit.....	4-6
4.3.2	How to Edit a Persistence Unit.....	4-6
4.4	Implementing Services with EJB Session Beans	4-7
4.4.1	How to Create a Session Bean.....	4-7
4.4.1.1	Generating Session Facade Methods	4-7
4.4.1.2	Choosing a Package.....	4-8
4.4.1.3	Remote, Local and Web Service Interfaces	4-9
4.4.2	What Happens When You Create a Session Bean	4-9
4.4.3	What You May Need to Know When Creating a Session Bean.....	4-11
4.4.4	How to Update an Existing Session Bean With New Entities.....	4-11
4.4.5	How to Add New Methods to the Session Bean.....	4-11
4.5	Accessing Entities from a Session Bean.....	4-12
4.5.1	How to Use the Entity Manager to Access Entity Data	4-13
4.5.1.1	Injecting an EntityManager Instance	4-13
4.5.1.2	Java SE Service Client Using an EntityManagerFactory.....	4-13
4.5.1.3	Looking up the EntityManager Through JNDI.....	4-14
4.5.2	How to Create a Named Query	4-14
4.5.3	How to Expose Named Queries On the Session Facade.....	4-14
4.5.3.1	How to Change the Return Type of a Method That Returns a Single Value....	4-15
4.6	Exposing a Session Bean as a Web Service.....	4-16
4.7	Exposing Services with ADF Data Controls	4-18
4.7.1	How to Create ADF Data Controls	4-18
4.7.2	Understanding the Data Control Files.....	4-18
4.7.2.1	About the DataControls.dcx File	4-19
4.7.2.2	About the Structure Definition Files.....	4-19
4.7.2.3	About the Entity XML Files	4-19
4.7.2.4	About the Design-time XML Files.....	4-19
4.7.3	Understanding the Data Control Palette.....	4-19
4.7.3.1	Overview of the Data Control Business Objects	4-21
4.7.3.2	Refreshing ADF Data Controls After Modifying Business Services	4-21
4.8	Testing EJBs	4-22
4.8.1	Testing Entities Outside the Container with a Java Service Facade.....	4-22
4.8.1.1	Create a Persistence Unit that Runs Outside the Container.....	4-22
4.8.1.2	Create a Java Service Facade.....	4-23
4.8.1.3	Run the Java Service Facade	4-24
4.8.2	Testing Inside the EE Container Using the Embedded Server	4-24
4.8.2.1	Create a Sample Java Client.....	4-24
4.8.2.2	Run the Service and Client.....	4-24
4.9	Deploying EJB Applications.....	4-25

5 Building Web Services

5.1	Introduction to Web Services and SOA	5-1
5.1.1	Simple Object Access Protocol	5-2
5.1.2	Web Services Description Language.....	5-3
5.1.3	Universal Description, Discovery, and Integration	5-4
5.2	Creating Web Services in JDeveloper	5-5
5.2.1	How Java and Primitive Types Are Serialized to XML	5-6
5.2.2	Web Service Development Tools in JDeveloper	5-6
5.2.3	How to Create a Bottom-Up Web Services	5-7
5.2.3.1	JAX-RPC and Oracle SOAP web services	5-8
5.2.3.2	Creating and deploying web services	5-8
5.2.4	Web Services metadata, JSR-181.....	5-10
5.2.5	How to Create a Web Service from EJB 3.0.....	5-11
5.2.6	How to Create a Top-Down Web Service	5-12
5.2.7	Deploying Web Services from JDeveloper.....	5-15
5.3	Using Web Services in JDeveloper	5-15
5.3.1	How to Create a Web Service Proxy	5-17
5.3.2	What Happens When You Create a Web Service Proxy	5-18
5.3.3	How to Create a Data Control From a Web Service	5-22
5.3.4	What Happens When You Create and Use a Web Service Data Control	5-22
5.4	Debugging, Testing and Analyzing Web Services in JDeveloper	5-23
5.4.1	Debugging Web Services in JDeveloper.....	5-24
5.4.1.1	Locally Debugging a Web Service	5-24
5.4.1.2	Remotely Debugging a Web Service.....	5-24
5.4.2	Analyzing Web Services for Interoperability	5-24
5.4.3	Using the HTTP Analyzer	5-26

6 Using Oracle Enterprise Service Bus

6.1	Introduction to Oracle Enterprise Service Bus	6-1
6.2	Creating an Oracle Enterprise Service Bus Flow	6-3
6.2.1	What Happens When you Create an Oracle Enterprise Service Bus Flow	6-4
6.2.2	What You May Need to Know.....	6-5
6.3	Creating Oracle Enterprise Service Bus Projects	6-6
6.3.1	How to Create an ESB Project	6-6
6.3.2	What Happens When You Create an ESB Project.....	6-6
6.4	Creating Oracle Enterprise Service Bus Systems or Service Groups.....	6-6
6.4.1	How to Create an ESB System or Service Group	6-7
6.4.2	What Happens When You Create an ESB System or Service Group	6-7
6.5	Browsing for Deployed Services Using the Service Explorer Browser.....	6-8
6.5.1	How to Browse for Deployed Services	6-8
6.6	Adding Endpoint Properties for Adapter or SOAP Services	6-9
6.6.1	How to Add Endpoint Properties	6-10

6.7	Adding SOAP Services to Oracle Enterprise Service Bus	6-11
6.7.1	How to Add SOAP Services.....	6-11
6.7.2	How to Modify SOAP Services.....	6-12
6.7.3	How to Delete SOAP Services	6-12
6.7.4	What Happens When You Add SOAP Services.....	6-12
6.7.5	What Happens at Runtime	6-13
6.8	Adding Adapter Services to Oracle Enterprise Service Bus	6-13
6.8.1	How to Add Adapter Services.....	6-14
6.8.2	How to Modify Adapter Services.....	6-15
6.8.3	How to Delete Adapter Services	6-15
6.8.4	What Happens When You Add Adapter Services.....	6-16
6.8.5	What Happens at Runtime	6-17
6.9	Adding Routing Services to Oracle Enterprise Service Bus	6-18
6.9.1	How to Add Routing Services	6-19
6.9.1.1	How to Generate the WSDL for a Routing Service from an Existing XSD File	6-20
6.9.1.2	How to Generate the WSDL to Create a Routing Service Based on a Sample File.....	6-21
6.9.1.3	How to Select an Existing WSDL to Create a Routing Service	6-22
6.9.2	How to Modify Routing Services	6-23
6.9.3	How to Delete Routing Service.....	6-24
6.9.4	What Happens When You Add Routing Services	6-24
6.9.5	What Happens at Runtime	6-25
6.10	Specifying Routing Rules.....	6-25
6.10.1	How to Access Routing Rules.....	6-25
6.10.2	How to Specify Target Operations.....	6-26
6.10.3	How to Specify the ESB Systems From which Messages are Accepted	6-27
6.10.4	How to Specify Synchronous or Asynchronous Execution.....	6-27
6.10.5	How to Specify Routing Rules Priority	6-27
6.10.6	How to Create an XSL File for Data Structure Transformation.....	6-28
6.10.7	How to Add An Expression for Filtering Messages Based on Payload	6-30
6.10.8	What Happens When You Specify Routing Rules.....	6-31
6.10.9	What Happens at Runtime	6-31
6.11	Adding a Domain-Value Map to Oracle Enterprise Service Bus	6-32
6.11.1	How to Create a New Domain-Value Map.....	6-32
6.11.2	How to Export a Domain-Value Map	6-34
6.11.3	How to Import a Domain-Value Map	6-35
6.11.4	How to Use a Domain-Value Map in a Data Transformation Map	6-35
6.12	Registering ESB Projects and Services with the ESB Server	6-38
6.12.1	How to Register ESB Services with the ESB Server	6-38
6.13	Testing and Validating the Oracle Enterprise Service Bus Flow	6-39
6.13.1	How to Test ESB Services Using Oracle Enterprise Manager	6-39
6.13.2	What Happens When You Test Using Oracle Enterprise Manager	6-42

7 Using Oracle BPEL Process Manager

7.1	Introduction to Oracle BPEL Process Manager	7-1
7.2	Creating an Application and a BPEL Project	7-4
7.2.1	How to Create an Application	7-5
7.2.2	How to Create a BPEL Project	7-6
7.2.3	What Happens When You Create an Application and a BPEL Project	7-6
7.3	Invoking a Web Service	7-10
7.3.1	How to Invoke a Web Service	7-11
7.3.1.1	Creating a Partner Link	7-12
7.3.1.2	Creating a Scope Activity	7-13
7.3.1.3	Creating an Invoke Activity Inside a Scope Activity	7-13
7.3.1.4	Creating an Initial Assign Activity Inside a Scope Activity	7-14
7.3.1.5	Creating a Second Assign Activity Inside a Scope Activity	7-15
7.3.1.6	Creating a Receive Activity	7-15
7.3.2	What Happens When You Invoke a Web Service	7-16
7.4	Using an Assign Activity	7-18
7.4.1	How to Use an Assign Activity	7-20
7.4.2	What Happens When You Use an Assign Activity	7-22
7.5	Creating a Transformation	7-22
7.5.1	How to Create a Transformation	7-25
7.5.2	What Happens When You Create a Transformation	7-27
7.6	Communicating with a Database	7-28
7.6.1	How to Configure a Database Adapter	7-29
7.6.2	What Happens When You Configure a Database Adapter	7-31
7.7	Creating Conditional Branching Using a Switch Activity	7-33
7.7.1	How to Create Conditional Branching Using a Switch Activity	7-35
7.7.2	What Happens When You Create Conditional Branching Using a Switch Activity	7-36
7.8	Using Business Rules and a Decide Activity	7-36
7.8.1	How to Use the Decision Service Wizard to Create a Decision Service Partner Link	7-37
7.8.2	How to Create a Decide Activity	7-40
7.8.3	What Happens When You Use Business Rules and a Decide Activity	7-40
7.9	Creating a Parallel Flow	7-43
7.9.1	How to Create a Parallel Flow	7-44
7.9.2	What Happens When You Create a Parallel Flow	7-44
7.10	Handling Faults	7-48
7.10.1	How to Handle Faults	7-50
7.10.1.1	Handling Faults with a Catch or CatchAll Branch	7-50
7.10.1.2	Handling Faults with a Throw Activity	7-51
7.10.2	What Happens When You Handle Faults	7-52
7.11	Using Sensors for Monitoring	7-54
7.11.1	How to Use Sensors for Monitoring	7-55
7.11.2	What Happens When You Use Sensors	7-56

7.12	Creating a Human Workflow Task	7-58
7.12.1	How to Create a Human Workflow Task	7-59
7.12.2	What Happens When You Create a Human Workflow Task	7-62
7.12.3	How to Assign Input and Output Parameters for the Human Workflow Task.....	7-63
7.12.4	How to Generate a Task Display Form	7-65
7.12.5	What Happens When You Generate a Task Display Form	7-65
7.12.6	How to Log In to the Worklist Application	7-66
7.12.7	What Happens at Runtime	7-67
7.13	Creating an Email Notification	7-67
7.13.1	How to Create an Email Notification.....	7-68
7.13.1.1	Adding an Email Service to a BPEL Process	7-69
7.13.1.2	Using the Identity Lookup Dialog	7-69
7.13.1.3	Using the XPath Expression Builder.....	7-71
7.13.1.4	Configuring Email Accounts	7-72
7.13.2	What Happens When You Create an Email Notification	7-74
7.14	Compiling and Deploying a BPEL Process	7-78
7.14.1	How to Compile and Deploy a BPEL Process	7-79
7.14.2	What Happens When You Compile and Deploy a BPEL Process.....	7-80
7.14.3	What Happens at Runtime	7-80

8 Creating and Using Rules for a Decision Service

8.1	Introduction to Oracle Business Rules and Rule Author	8-1
8.1.1	What Are Business Rules?	8-2
8.1.2	Oracle Business Rules Terminology.....	8-3
8.1.3	Using Existing Rules and Facts.....	8-3
8.2	Creating a Rule Author User.....	8-5
8.2.1	How to Create a Rule Author User with Enterprise Manager.....	8-5
8.3	Using Rule Author	8-6
8.3.1	How to Start Rule Author.....	8-6
8.4	Creating and Saving a Dictionary with Rule Author	8-7
8.4.1	How to Set Up a File Repository	8-7
8.4.2	How to Set Up a WebDAV Repository.....	8-8
8.4.3	How to Create and Connect to a Repository	8-9
8.4.4	How to Create and Save a Rule Author Dictionary	8-10
8.4.5	What Happens When You Save a Dictionary.....	8-11
8.5	Importing an XML Schema into the Data Model with Rule Author.....	8-11
8.5.1	How to Import XML Schema into the Data Model.....	8-11
8.5.1.1	Performing XML Schema Processing	8-11
8.5.1.2	Importing XML Schema Elements into the Data Model.....	8-14
8.5.1.3	Saving the Dictionary.....	8-15
8.5.2	What Happens When You Import XML Schema into the Business Model	8-15
8.5.3	What You May Need to Know When You Import XML Schema.....	8-16
8.6	Defining a Variable in the Data Model with Rule Author.....	8-16
8.6.1	How to Define a Business Rules Variable in the Data Model	8-16

8.7	Creating Rules with Rule Author	8-17
8.7.1	How to Create a Rule Set.....	8-17
8.7.2	How to Create a Rule	8-18
8.7.2.1	Add a Pattern for the Rule	8-20
8.7.2.2	Define a Test for the Rule	8-21
8.7.2.3	Add an Action for a Rule.....	8-23
8.7.3	What Happens When You Define a Ruleset and Rules	8-24
8.7.4	What You May Need to Know When You Define a Ruleset and Rules	8-24

9 Developing a Web Application in an SOA System

9.1	Introduction to Developing a Web Application in an SOA System	9-1
9.1.1	View Layer Technologies Supported	9-2
9.1.2	Business Services Technologies Supported by ADF Model	9-2
9.1.3	Declarative Development with Oracle ADF and JavaServer Faces.....	9-3
9.1.3.1	Declarative Page Navigation with JSF.....	9-5
9.1.3.2	Declarative Data Binding with Oracle ADF Model Layer.....	9-5
9.1.4	Building a Web Application.....	9-7
9.2	Creating a Web Application Workspace	9-8
9.2.1	How to Create an Application Workspace	9-8
9.2.2	What Happens When You Use an Application Template to Create a Workspace	9-8
9.3	Creating a Web Page	9-9
9.3.1	How to Create a JSF Web Page	9-9
9.3.2	What Happens When You Create a JSF Web Page.....	9-11
9.3.3	What You May Need to Know About Using the JSF Navigation Diagram	9-12
9.4	Creating and Using a Backing Bean for a Web Page	9-12
9.4.1	How to Create and Configure a Backing Bean.....	9-13
9.4.2	What Happens When You Create and Configure a Backing Bean.....	9-14
9.4.3	How to Use the Automatic Component Binding Feature	9-14
9.4.4	What Happens When You Use Automatic Component Binding in JDeveloper	9-15
9.5	Displaying Collection and Detail Data Using ADF Data Binding.....	9-16
9.5.1	How to Create Navigation Between Pages	9-17
9.5.2	What Happens When You Create Navigation	9-18
9.5.3	What Happens at Runtime	9-19
9.5.4	What You May Need to Know About the Navigation Modeler.....	9-19
9.5.5	How to Create a Table to Display Collection Data on a Page	9-20
9.5.6	What Happens When You Add a Collection to a Page.....	9-23
9.5.7	What Happens at Runtime	9-25
9.5.8	How to Display Details on a Page.....	9-25
9.5.9	What Happens When You Display Detail Data	9-26
9.6	Collecting Data for Use by a Web Service	9-28
9.6.1	How to Create an Input Form Using a Web Service Data Control	9-28
9.6.2	What Happens When You Create an ADF Input Form	9-29
9.6.3	How to Invoke a Service Using a Backing Bean.....	9-30
9.6.4	What Happens When You Invoke a Service From a Web Page.....	9-33
9.7	Testing the Web Application.....	9-35

10 Securing the System

10.1	Introduction to Oracle Web Services Manager.....	10-2
10.2	Securing Web Services Using Oracle Web Services Manager	10-4
10.2.1	When to Use Agents or Gateways.....	10-4
10.2.2	How Oracle Web Services Manager Uses Policies.....	10-4
10.2.3	Security Architecture.....	10-5
10.3	Authenticating Users with an Oracle Web Services Manager Server Agent	10-6
10.3.1	How to Register a Server Agent	10-7
10.3.2	What Happens When You Register an Agent	10-8
10.3.3	How to Define the Policy Set for the Server Agent	10-8
10.3.4	What Happens When You Define the Policy Set for the Agent.....	10-13
10.3.5	How to Set Up a File with User Credentials	10-13
10.3.6	How to Install the Server Agent	10-14
10.3.7	How to Configure the Web Services Agent in Application Server Control.....	10-15
10.3.8	What Happens When You Configure the Web Services Agent in Application Server Control	10-17
10.3.9	How to Test the Authentication	10-17
10.3.10	What Happens at Runtime	10-19
10.3.11	How to Configure Oracle BPEL Process Manager to Send a Username Token	10-20
10.4	Encryption with an Oracle Web Services Manager Gateway	10-20
10.4.1	How to Register the Gateway	10-23
10.4.2	What Happens When You Register a Gateway	10-25
10.4.3	How to Locate the Service WSDL	10-25
10.4.4	How to Register the Web Service at the Gateway	10-27
10.4.5	How to Redirect Clients to the Gateway	10-29
10.4.6	How to Create the Certificate Keystore.....	10-31
10.4.7	What You May Need to Know About Certificate Keystores.....	10-32
10.4.8	How to Define Encryption and Decryption Policy Steps	10-32
10.4.9	How to Test Message Encryption.....	10-35

11 Deploying SOA Applications with Enterprise Manager

11.1	Introduction to Deploying SOA Applications with Oracle Enterprise Manager Application Server Control.....	11-1
11.2	Preparing for Deployment	11-2
11.2.1	How to Define Deployment Requirements	11-2
11.2.2	How to Configure Load Balancing Within a Cluster	11-2
11.2.2.1	OC4J Groups	11-2
11.2.2.2	Multiple JVMs	11-3
11.2.2.3	Application Clustering (State Replication)	11-3
11.2.3	How to Create Data Sources	11-4
11.2.4	How to Deploy and Configure Standalone Resource Adapters.....	11-5
11.3	Deploying the Application with Oracle Enterprise Manager Application Server Control.....	11-6
11.3.1	How to Deploy the Application	11-7
11.3.2	What Happens When You Deploy the Application	11-11
11.3.3	How to Deploy to OC4J Instances Within a Cluster.....	11-12

11.4	Performing Post-Deployment Tasks	11-13
11.4.1	How to Set Up Notifications	11-13
11.4.2	How to Redeploy the Application	11-13
11.4.2.1	What Happens When You Redeploy the Application	11-13
11.4.2.2	Impact of Redeploying a Parent Application	11-14
11.4.3	How to Undeploy the Application.....	11-14
11.4.3.1	What Happens When You Undeploy the Application	11-14
11.4.3.2	Impact of Undeploying a Parent Applications	11-14
11.4.4	Troubleshooting Deployment Errors	11-15

Part III Monitoring SOA Systems

12 Monitoring Oracle Enterprise Service Bus

12.1	Introduction to Monitoring with Oracle ESB Control	12-1
12.2	Components of Oracle ESB Control	12-2
12.2.1	How to Open the Oracle ESB Control	12-2
12.3	Using the Services View to Review and Update ESB Systems.....	12-3
12.3.1	How to View and Modify System Definitions	12-3
12.4	Using the Services View to Review and Update ESB Services.....	12-4
12.4.1	How to View and Modify Service Definitions	12-6
12.5	Using the Maps View to Define Domain-Value Maps	12-7
12.6	Using the Instances View to Monitor Message Instances	12-7
12.6.1	How to View All Message Instances.....	12-8
12.7	Searching for Message Instances	12-9
12.7.1	How to Search for Message Instances.....	12-10
12.8	Managing Message Instance Tracking.....	12-10
12.8.1	How to Enable and Disable Message Instance Tracking	12-10
12.8.2	How to Purge Message Instances.....	12-11
12.9	Defining Trackable Fields	12-11
12.9.1	How to Define Trackable Fields	12-12
12.9.2	How to Enable or Disable Trackable Fields.....	12-13
12.9.3	What Happens When You Define a Trackable Field.....	12-13
12.9.4	What Happens at Runtime	12-14
12.10	Viewing Log Files	12-14
12.10.1	How to View Log Files.....	12-14
12.11	Error Handling in Message Instance Processing.....	12-15
12.11.1	About Adapter Error Handling.....	12-16
12.11.2	How to Resubmit Messages on Errors.....	12-16
12.11.3	What Happens When You Resubmit Messages on Errors	12-18

13 Monitoring Oracle BPEL Process Manager

13.1	Introduction to Oracle BPEL Control.....	13-1
13.2	Components of Oracle BPEL Control	13-2
13.2.1	How to Log In to Oracle BPEL Control.....	13-2
13.3	Using the Dashboard Tab to Monitor Deployed, Running, and Completed Processes	13-2
13.3.1	How to Monitor Deployed, Running, and Completed Processes	13-3

13.4	Using the BPEL Processes Tab to Manage a Process Life Cycle	13-4
13.4.1	How to Manage a Process Life Cycle.....	13-5
13.4.2	How to Manage a Specific Process.....	13-6
13.5	Using the Instances Tab to Monitor Process Instances.....	13-8
13.5.1	How to Monitor Process Instances	13-8
13.6	Using the Activities Tab to Monitor Process Activities.....	13-9
13.7	Managing BPEL Domains.....	13-9
13.7.1	How to Create BPEL Domains.....	13-10

14 Monitoring Security

14.1	Introduction	14-1
14.2	Monitoring Security	14-1
14.2.1	The Oracle Web Services Manager Monitor	14-2
14.2.2	Execution Metrics	14-2
14.2.3	Security Statistics	14-3
14.2.4	Service Statistics	14-3
14.2.5	Alarms	14-3
14.2.6	Role Management.....	14-3

15 Monitoring SOA Applications with Enterprise Manager

15.1	Introduction to Monitoring the OC4J Environment	15-1
15.1.1	How to Monitor an OC4J Instance	15-1
15.1.2	How to Monitor the Cluster Topology	15-3
15.1.3	How to Monitor Deployed Applications	15-4
15.1.4	How to Monitor Web Modules and EJB Module Performance	15-4
15.1.5	How to Monitor JVM Metrics	15-5
15.1.5.1	Monitoring JVM Metrics in the Oracle Application Server Environment	15-5
15.1.5.2	Monitoring J2SE 5.0 JVM Metrics.....	15-5
15.2	Using the Adobe SVG Viewer to View Performance Charts.....	15-6

Index

Preface

Welcome to the Oracle SOA Suite Developers Guide!

Audience

This document is intended for developers and designers working on applications built on service-oriented architecture (SOA) using the components of Oracle SOA Suite, including, Oracle JDeveloper, Oracle Process Manager, Oracle BPEL Process Manager, Oracle BPEL Control, Oracle Enterprise Service Bus, Oracle ESB Control, Oracle Web Services Manager, and Oracle Enterprise Manager Application Server Control.

Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible, with good usability, to the disabled community. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Accessibility standards will continue to evolve over time, and Oracle is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For more information, visit the Oracle Accessibility Program Web site at

<http://www.oracle.com/accessibility/>

Accessibility of Code Examples in Documentation

Screen readers may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, some screen readers may not always read a line of text that consists solely of a bracket or brace.

Accessibility of Links to External Web Sites in Documentation

This documentation may contain links to Web sites of other companies or organizations that Oracle does not own or control. Oracle neither evaluates nor makes any representations regarding the accessibility of these Web sites.

TTY Access to Oracle Support Services

Oracle provides dedicated Text Telephone (TTY) access to Oracle Support Services within the United States of America 24 hours a day, seven days a week. For TTY support, call 800.446.2398.

Related Documents

For more information, see the following documents:

Oracle JDeveloper and Oracle Application Development Framework:

- *Oracle Application Development Framework Developer's Guide*
- *Oracle Application Developer Framework Developer's Guide for Forms/4GL Developers*
- *Oracle JDeveloper 10g Online Help*

Oracle Enterprise Service Bus:

- *Oracle Enterprise Service Bus Developer's Guide*
- *Oracle Enterprise Service Bus Installation Guide*
- *Oracle Enterprise Service Bus Quick Start Guide*

Oracle BPEL Process Manager and Oracle BPEL Control:

- *Oracle BPEL Process Manager Administrator's Guide*
- *Oracle BPEL Process Manager Developer's Guide*
- *Oracle BPEL Process Manager Installation Guide*
- *Oracle BPEL Process Manager Order Booking Tutorial*
- *Oracle BPEL Process Manager Quick Start Guide*

Oracle Business Rules:

- *Oracle Business Rules User's Guide*
- *Oracle Business Rules Language Reference Guide*
- *Oracle Business Rules Java API Reference*

Oracle Web Services Manager:

- *Oracle Web Services Manager Administrator's Guide*
- *Oracle Web Services Manager Deployment Guide*

Oracle Application Server

- *Oracle Application Server Administrator's Guide*
- *Oracle Application Server Tutorial*
- *Oracle Application Server Performance Guide*
- *Oracle Application Server Adapter Concepts*
- *Oracle Application Server Adapter for Oracle Applications User's Guide*
- *Oracle Application Server Adapter for Files, FTP, Databases, and Enterprise Messaging User's Guide*
- *Oracle Containers for J2EE Configuration and Administration Guide*
- *Oracle HTTP Server Administrator's Guide*

Conventions

The following text conventions are used in this document:

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

Part I

Getting Started With Oracle SOA Suite

Part I contains the following chapters:

- [Chapter 1, "Introduction to SOA and the Oracle SOA Suite"](#)
- [Chapter 2, "The SOA Order Booking Application"](#)

Introduction to SOA and the Oracle SOA Suite

This chapter provides an overview of the benefits of creating applications based on a service-oriented architecture (SOA) and how you can use the components included in the Oracle SOA Suite to design, develop, deploy and monitor an SOA application.

This chapter includes the following sections:

- [Section 1.1, "Overview of SOA and the Oracle SOA Suite"](#)
- [Section 1.2, "SOA Benefits"](#)
- [Section 1.3, "Using Oracle SOA Suite to Adopt SOA"](#)

1.1 Overview of SOA and the Oracle SOA Suite

Changing markets, increasing competitive pressures and evolving customer needs are placing greater pressure on IT to deliver greater flexibility and speed. Today every organization is faced with the need to predict change in a global business environment, to rapidly respond to competitors, and to best exploit organizational assets for growth. In response to these challenges, leading companies are adopting service-oriented architecture (SOA) as a means of delivering on these requirements by overcoming the complexity of their application and IT environments. SOA represents a fundamental shift in the way new applications are designed, developed, and integrated with legacy business applications, and facilitates the development of enterprise applications as modular business services that can be easily integrated and reused.

Oracle SOA Suite is a standards-based best of breed suite that enables you to build service-oriented applications and deploy them to your choice of middleware platform. Oracle SOA Suite consists of the following:

- Integrated service environment (ISE) to develop services
- Multi-protocol enterprise service bus (ESB) to integrate applications
- Business process execution language (BPEL) -based orchestration engine to tie services into business processes
- Business rules engine to enable business policies to be captured and automated
- Web services management and security solution to enforce authentication and authorization policies on services and to monitor services and processes for compliance to SLAs
- Business Activity Monitoring (BAM) solution to gain real-time visibility into business entities and their interactions, and enables services to be optimized

- Enterprise Portal for employees, customers and partners to access content, access relevance performance metrics, collaborate and take actions via interaction with business processes
- Services registry for discovering and managing the lifecycle of services

Read this chapter to understand:

- Key benefits of SOA
- Key benefits of Oracle SOA Suite

1.2 SOA Benefits

Increasingly fragmented and complex infrastructures are limiting IT's ability to deliver on business needs. Many organizations have inherited disjointed legacy systems and packaged applications, a large proportion of which were never designed for information interoperability, integration, and reuse. The result of this is that most of the IT budget goes into maintenance of the current IT infrastructure and only a small amount is available for new functionalities to drive new business opportunities. The major portion of budget for new capabilities goes into the cost of integrating new functionality into the existing systems that offers poor support for interoperability.

Traditionally, business information systems have been developed with a functional orientation often resulting in silos of services and information. The fundamental problem with this is that end-to-end business processes, which must span silos, are not adaptable to change as business needs evolve. The processes become fragmented and embedded deep within systems. Enterprise application integration (EAI) and other traditional middleware solutions partially address this by enabling systems to communicate with each other, but they don't fully solve the problem. Their ability to create cross application business processes is inadequate, and they allow only limited business process adaptability. Moreover, these traditional solutions come at a high cost. The majority of EAI and traditional middleware solutions use proprietary technology, which causes dependency on specialized skills and to that single vendor's products. In addition, the systems become tightly coupled, so if an interface change occurs in one system, all other systems need to be adjusted. So not only does the technology make it difficult to make changes; it can become cost-prohibitive to do so.

SOA helps address the fragmented IT landscape and addresses the difficulties associated with silos of IT infrastructure and applications. It enables greater flexibility through:

- Greater interoperability: SOA, and the industry standards underpinning it, enable existing siloed applications to interoperate seamlessly and in an easier to maintain manner than any traditional EAI solution.
- Increased reuse: Once legacy systems and applications are service enabled, these services can be reused, which results in reduced ongoing development costs and results in reduced time to market. Further, business processes built as an orchestration of services can also be exposed as services, further increasing reuse.
- More agile business processes: SOA reduces the gap between the business process model and implementation. This enables changes to business processes already implemented as orchestrations of services to be easily captured and implemented.

- Improved visibility: SOA can give improved business visibility by enabling business capabilities exposed as services, and the status of in-flight business processes automated with Business process management (BPM) technology, to be rapidly integrated into service-enabled enterprise portals aiding business decision-making.
- Reduced maintenance costs: SOA development encourages duplicated overlapping business capabilities (services) that span multiple applications and systems to be consolidated into a small number of shared services. This enables elimination of redundant services and reduces the cost of maintaining systems by providing a single point of change for application logic. Further, SOA gives IT the means to gradually phase out legacy systems and applications while minimizing disruption to the applications that are built on or are integrated with them using SOA principles. This frees up funds for new projects.
- Compliance and governance: By realizing better and more standardized operational procedures, SOA provides the basis for a comprehensive security solution, and enables better visibility into business operations and exception conditions.

ISEs enable developers to expose application functionality as services. ESB technology makes service connections less brittle when changes need to be made. For example, ESB supports a logical naming construct and decouples applications from protocols and data formats. BPM solutions such as those based on BPEL enable services to be orchestrated into business processes. Processes built using a BPM solution can be reused, changed easily in response to business requirements, and enable real-time process visibility. Business activity monitoring solutions realize monitoring of KPIs and SLAs and enable business entities to take preemptive actions.

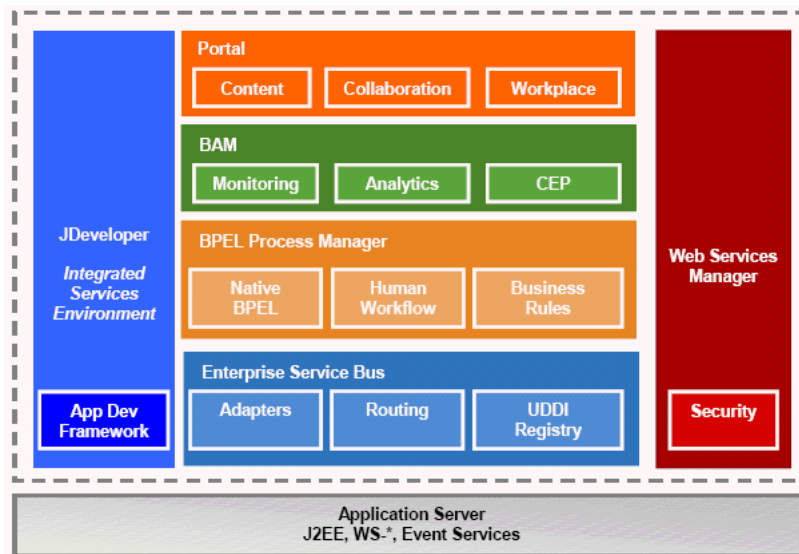
1.3 Using Oracle SOA Suite to Adopt SOA

Oracle SOA Suite provides a comprehensive suite of key best of breed SOA technologies that plugs into heterogeneous IT infrastructures and enables enterprises to incrementally adopt SOA. The components of the suite benefit from common capabilities including a single deployment and management model and tooling, end-to-end security, and unified metadata management. Oracle SOA Suite is unique in that it provides the following set of integrated capabilities:

- Messaging
- Service discovery
- Orchestration / BPM
- Activity monitoring
- Web services management and security
- Business rules
- Services directory and development tool
- Service-enabled portal

At the same time, Oracle SOA Suite provides support for existing middleware technologies, such as third party J2EE application servers, development tools, message queues, and ESBs. Hence, enterprise IT departments can adopt the whole suite, which benefits from an integrated set of capabilities, or, adopt pieces individually.

The components of Oracle SOA Suite are shown in [Figure 1-1](#).

Figure 1–1 Oracle SOA Suite

1.3.1 Creating Services

Oracle JDeveloper, Oracle Application Development Framework (Oracle ADF), and Oracle TopLink are development components of Oracle SOA Suite that together form a comprehensive ISE to develop, compose, and orchestrate services into business processes that can be deployed, registered, and consumed from several types of user interfaces, including desktop clients, browsers, and mobile and telnet devices.

Oracle JDeveloper enables developers to model, create, discover, assemble, orchestrate, test, deploy, and maintain composite applications based on services. Oracle JDeveloper supports SOA principles and XML Web services standards, as well as traditional Java, J2EE, and PL/SQL component and modular code mechanisms.

Oracle ADF is a model-driven SOA framework that automates and manages business and data services and provides a standard data-and service binding layer based on the JSR 227 specification. This binding layer can be used with process flows, page flows, and service invocations. Oracle ADF also implements SOA design practices, and makes user interfaces as loosely coupled as services themselves.

Oracle TopLink is a data services framework that allows access to relational and XML data. It provides visual mapping tools for facilitating object to relational and object to XML mapping. Oracle TopLink and Oracle ADF frameworks simplify creation of business and data services that can be invoked from rich web interfaces of service-oriented applications.

Oracle JDeveloper, Oracle ADF, and Oracle TopLink together provide a comprehensive integrated services environment and framework that allows developers to build model-driven applications and business processes which can then be deployed and registered as applications, services, or business processes into Oracle SOA Suite components.

Key capabilities include:

- **Creating business services:** Oracle JDeveloper supports web services invocation framework (WSIF) bindings that allow for the publishing of Java, Enterprise Java Beans (EJB), Java Message Service (JMS), and enterprise applications that are connected using Java Connector Architecture (JCA) adapters, as natively invoked services. Oracle JDeveloper also features support for REST (Representational State Transfer)-based services and for developing J2EE 1.4-compliant Web services, with the ability to create JAXRPC clients and services, and WS-Security, WS-Reliability, and WSMangement. Support for developing bottom-up services using metadata tags and contract-driven development using the WSDL editor is also included.
- **Creating business and user interface logic:** Oracle JDeveloper includes support for EJB 3.0 that simplifies back end business logic and persistence mapping aspect. Oracle JDeveloper wizards generate EJB 3.0 components, and property editors support the incremental development of business methods and mapping annotations. Support for building web applications using JavaServer Faces (JSF) technology enables rich user interfaces. Oracle JDeveloper provides several tools such as visual UI component editing using JSF WYSIWYG editors, drag-and-drop capabilities for JSF components onto pages, and visual diagrammer for rapid development of JSF navigation.
- **Binding user interfaces to services:** Oracle JDeveloper includes support for the JSR 227 specification that facilitates binding user interfaces to data sources and services. Support for drag-and-drop attributes, collections and operations from the Data Control Palette that exposes back end business systems, and for services and business process is also included. Also featured is support for data-binding Oracle ADF Faces components to back end business services and business processes.

1.3.2 Using an Enterprise Service Bus

As the “glue” for the enterprise application infrastructure, your integration platform provides the basis for gathering information that drives the organization from all types of data sources. Oracle ESB is the realization of the SOA and event driven architectures (EDA) whereby distributed applications are integrated in a loosely coupled paradigm. At their core, ESBs implement messaging to enable services to be integrated in a message-based paradigm: both synchronous and asynchronous styles. They also incorporate routing so that messages can be routed to the appropriate services based on rules governing both the message content and any external factors. Thirdly, ESBs also embody message transformation. Since ESBs enable routing and transformation logic to be changed at runtime, they enable more maintainable applications to be built, since service connections are less brittle.

Key capabilities for Oracle ESB include:

- **Reliable multi-transport bus:** Oracle ESB provides a flexible real-time enterprise backbone capable of supporting industry standard protocols such as SOAP, HTTP(s), or JMS. A special in-memory optimization is automatically used for service calls within the same virtual machine. It provides fast, scalable, guaranteed once-and-only-once message delivery using both point-to-point and publish/subscribe patterns. Oracle ESB can use Oracle's own JMS or Oracle Advanced Queuing (AQ) as a message transport; it is also certified with other messaging providers such as IBM MQ, SonicMQ, and Tibco.

- **Complex business data transformations:** Businesses require flexibility in combining data models from disparate systems. Oracle ESB utilizes standards-based data mapper functionality within Oracle JDeveloper to create transformation templates in the XSLT language for reuse across the enterprise. The auto-mapping feature increases user productivity by remembering and reusing common mappings from previous transformations.
- **Comprehensive management and deployment infrastructure:** At design time, Oracle ESB allows you to create virtual service names into lookup repositories, such as UDDI, that are later bound to real or physical application URLs defined during deployment. Oracle ESB Control allow you to build and visualize relationships between services and graph dependency charts or impact analysis for proposed changes to your systems. The viewer includes an ESB-wide search facility to locate components such as adapters, messages and active process instances based on unlimited input criteria. Centralized management of distributed applications is a key component of Oracle ESB.
- **Pervasive enterprise system connectivity:** Adapters provide key connectivity and discovery into enterprise and legacy system metadata to enable mapping of objects to real-time events in your ESB. Oracle provides built-in adapter wizards for Database, Oracle AQ, JMS, email, FTP and files, as well as enterprise applications such as the Oracle E-Business Suite, PeopleSoft, JDEdwards, SAP and legacy systems such as CICS, IMS and even TPF. All adapters conform to the J2EE Connector Architecture (JCA) open standard adopted by all major integration vendors.
- **Flexible content based routing:** The ability to filter and route data based on message content is critical to optimal management of your ESB. Oracle ESB enables routing in design time deployment descriptor definitions that can be modified at runtime for adjusting application efficiency. This minimizes the overhead of redeployment. For example, as system demand increases and you add servers to your cluster, you can dynamically route traffic based on content such as currency, region, product name, or any other contextual data. Oracle ESB supports a variety of rules capabilities for filtering including Oracle BAM, Oracle Business Rules and external providers. Content filtering can also be implemented in messaging systems such as JMS using configurable filter based subscriptions and message selectors.

1.3.3 Composing and Orchestrating Services

Oracle BPEL Process Manager enables business processes to be modeled, automated, and monitored. Unlike code-generation techniques for automating business processes, Oracle BPEL Process Manager includes a native BPEL engine that executes the processes. This approach not only enables reuse, but also enables visibility into in-flight business processes at the individual and aggregate levels (the latter being provided by Oracle BAM), and lays the foundation for close-loop business process management, process improvement, and compliance.

Oracle BPEL Process Manager provides a comprehensive, standards-based and easy to use solution for creating, deploying, and managing cross-application business processes with both automated and human workflow steps. It provides high-performance, reliable execution of service-oriented business processes defined with the BPEL standard. Its native support for standards such as BPEL, XML, XSLT, XPATH, JMS, JCA and web services makes it an ideal solution for creating integrated business processes that are truly portable across platforms. It also provides audit trails for both completed and in-flight processes, and process history that enables process improvement.

Finally, the Oracle BPEL Process Manager is a 100% native BPEL engine that coexists happily with existing middleware technologies and platforms, and provides an unrivaled process portability and vendor flexibility.

Key capabilities of Oracle BPEL Process Manager include:

- Rich tooling for integration: Oracle BPEL Process Designer within the Oracle JDeveloper environment, provides a unified design time environment to develop user interfaces and orchestration services. Built-in integration services enable developers to easily leverage advanced workflow, connectivity, and transformation capabilities from standard BPEL processes. These capabilities include support for XSLT and XQuery transformation as well as bindings to hundreds of legacy systems through JCA adapters and native protocols using WSIF. The extensible WSDL binding framework enables connectivity to protocols and message formats other than SOAP. Bindings are available for JMS, email, JCA, HTTPGET, POST, and many other protocols enabling simple connectivity to hundreds of back-end systems. This approach gives unparalleled performance, while maintaining ease of development. User-friendly wizards to set up simple and complex human workflow steps, configure adapters, and define complex transformation maps are provided as standard services. Human workflow services such as task management, notification management, and identity management are provided as built-in BPEL services to enable the integration of people and manual tasks into BPEL flows.
- Comprehensive monitoring and management: Oracle BPEL Control provides a user-friendly web-based interface for management, administration, and debugging of processes deployed to the BPEL server. Audit trails and process history/reporting information are automatically maintained and available through the Oracle BPEL Control. The workflow task lists and historical process analysis reports are also integrated into the same console.
- Un-paralleled scalability and availability: The core BPEL engine provides the most mature, scalable, and robust implementation of a BPEL server available today. The Oracle BPEL Process Manager executes standard BPEL processes and provides a “dehydration” capability so that the state of long-running flows is automatically maintained in a database, enabling clustering for both fail-over and scalability. The BPEL server leverages Oracle Containers for J2EE as an underlying J2EE application server, but also supports most major commercial application servers such as BEA WebLogic and JBoss.

1.3.4 Automating Business Policies with Business Rules

Agility is one of the biggest promises of SOA and BPM: the ability to make rapid changes to processes in step with the changes that occur inside of your business. Such changes are not always changes to the process. Often they are changes to the rules that drive the process. A typical business process often includes a number of decision points. These decision points generally have an effect on the process flow; for example, someone's credit rating may determine whether he/she requires manual approval of an order. These decisions are evaluated based on certain conditions and facts, which may be internal or external to the business process, and predefined company policies or rules. Oracle Business Rules Engine allows architects to easily define, automate, manage, and update the decision logic that directs enterprise applications from a single location without needing to write code or change the business processes calling them.

Rules engines are naturally of interest to enterprise architects building out SOAs, since they contribute to agility by enabling reduced time to automate, easier change, and easier maintenance for business policies and rules.

BPM technology and rules engines naturally fit together: BPM enables automated and flexible business processes; rules engines enable automated and flexible business policies.

Key capabilities of Oracle Business Rules:

- Capturing business policies across all applications: In the past, rules engines were primarily used as a technology to solve highly complex problems requiring a great degree of inferencing. More recently, the rules market has evolved such that rules are now being used for the implementation of business policies. Oracle Business Rules enables business policies to be abstracted out of any application, not just applications that are designed to SOA. It is based on the Jess Rules Engine, but conforms to the JSR-94 specification and enables facts to be defined in Java.
- Automation of business policies in business processes: Business processes typically involve decision steps. Oracle Business Rules enables decisions and business policies to be abstracted out of the business process, providing a single location for policy management and change. Oracle Business Rules enables facts external to the business process to be asserted in the evaluation of business rules, unlike monolithic BPM Suites. The rules are separated from the process logic and can be changed independently without impacting them. This makes business processes less brittle; rule changes can be deployed without modifying or rebuilding rule-enabled processes and applications.

1.3.5 Securing and Managing Services Using Oracle Web Services Manager

Oracle Web Services Manager (Oracle WSM) enables IT to effectively monitor, manage and secure services and interactions between these services in an SOA. It provides tools for building security and operations policies that can be layered over new or existing applications and web services; runtime facilities for intercepting calls to and from an application or service and then executing these policies; dashboards for monitoring these policies as they execute, to ensure service levels and potential problems; and, alerting to enable corrective actions to be taken in a timely fashion.

Oracle WSM can centrally define policies that govern web services operations such as access policy, logging policy, and content validation, and then wrap these policies around services, with no modification to existing web services being required. Also, Oracle WSM collects monitoring statistics to ensure service levels and security, and displays them in a web dashboard. As a result, Oracle WSM brings enterprises better control and visibility over web services.

Key capabilities for Oracle WSM include:

- Policy management: The Oracle WSM Policy Manager is a graphical tool for building new security and operations policies, storing policies and managing distribution and updates to runtime agents and gateways. Policy Manager supports both client-side and service-side policy enforcement, and allows administrators to configure operational rules and propagate them to the appropriate enforcement components across an application deployment of any scale and complexity. Oracle WSM has out-of-the-box support for authentication and authorization using HTTP basic authentication, COREid, Netegrity, LDAP, and X.509 Certificates. It leverages Oracle COREid, LDAP, and Netegrity for role-based invocation access, and supports Security Assertion Markup Language (SAML) for interoperability between different security systems.

- **Enforcement** - To ensure maximum deployment flexibility, Oracle WSM provides two kinds of enforcement components: policy gateways and policy agents. Policy gateways are deployed in front of a group of applications or services. Policy gateways can intercept inbound requests to these applications in order to enforce policy steps, adding application security and other operation rules to applications that are already deployed. Policy agents provide an additional, fine-grained level of security by plugging directly into an application or service. Oracle WSM enforces true end-to-end message-level security, supports WS-Security for authentication and message-level security including encryption and signing, and supports full and partial encryption and decryption step.
- **Monitoring**: Oracle Web Services Manager Monitor collects data from gateways and agents as they execute policies and display results in a graphical format. This allows administrators to set quality of service levels for each application. The monitor displays alerts when the application exceeds established targets. It also provides IT operations staff with real-time visibility into the health, performance, security and utilization of crucial web services, including end-to-end monitoring of business processes. By harnessing the real-time data collection capabilities of the enforcement components, the monitor enables administrators to analyze discrepancies between expected and actual performance and to monitor compliance with IT operational best practices in real-time, such as audits on security violations on a per-web service, per-operation and per-client basis. The result is best-practice security and operations across all applications and services, and across an enterprise and its partners, regardless of the how these applications and services were developed.
- **Comprehensive support for protocols and third-party platforms**: Oracle WSM works with multiple web services platforms, and providers including BEA Systems, IBM, Microsoft, Netegrity, TIBCO, and VeriSign. For example, sites using TIBCO BusinessWorks can install Oracle WSM agents to act as SOAP interceptors that enforce web service policies. Oracle WSM provides out-of-the-box, native support for multiple transports, including HTTP, HTTPS, JMS, and IBM WebSphereMQ and multiple messaging models, including synchronous and asynchronous messaging. Furthermore, it provides content-based routing and built-in failure handling, including message queuing, fail-over routing, and configurable message retry capabilities.

1.3.6 Deploying SOA Applications

The Oracle SOA Suite deploys to leading application servers as well as Oracle Application Server 10g Release 3 (10.1.3.1.0), which includes several new features to support very high performance, scalability and high availability in a grid computing architecture. Dynamic workload management, enhanced clustering, automated back up and recovery, automated disaster recovery, and a new JMX-based management console are some of the new features designed to make it one of the industry's only grid-ready middleware solutions.

Oracle Application Server is certified with open source software including Spring, Apache Struts, Apache Axis, Apache MyFaces, Hibernate, Tapestry, JUnit, CVS, SubVersion, Ant, Eclipse and Log4J. Oracle Application Server 10g features support for a host of standards required to build next-generation SOAs, including WS-Reliable Messaging, WS-Security, WS-Federation, Web Services Metadata, WSIF and REST web services.

1.3.7 Benefits of Oracle SOA Suite

Oracle SOA Suite can help you achieve the best total value of opportunity by means of:

- **Develop and deploy applications faster:** As the market's only integrated SOA suite, Oracle SOA Suite greatly reduces the overall cost of architecting, developing, deploying, and managing applications. Applications are built faster, they can be put into production sooner, and their associated benefits can be realized sooner. You save money, and you can reassign software engineers and funds to other projects
- **Reduce application deployment costs:** Oracle SOA Suite is the only SOA suite designed to leverage grid computing to lower costs by deploying enterprise applications on modular, low-cost hardware and storage.
- **Reduce maintenance and management costs:** Oracle SOA Suite lowers management costs by automating software provisioning across groups of systems and by centralizing systems monitoring and administration. It lowers security administration costs by centralizing identity and access management.

Oracle SOA Suite can help you achieve the best total value of opportunity by reducing your development costs, by reducing your deployment costs, and by reducing ongoing maintenance costs, thus providing you with the fastest return on your investment.

The SOA Order Booking Application

As a companion to this guide, the SOA Order Booking application was created to demonstrate how a number of applications, both internal to an enterprise and external at other sites, can be integrated using the SOA paradigm and Oracle SOA Suite to create one cohesive ordering system. It is used as an example throughout this guide to illustrate points and provide code samples.

Before examining the individual components and their source code in depth, you may find it helpful to install and become familiar with the functionality of the SOA Order Booking application.

This chapter includes the following sections:

- [Section 2.1, "Introduction to the SOA Order Booking Application"](#)
- [Section 2.2, "Installing the SOA Order Booking Demonstration Application"](#)
- [Section 2.3, "An Overview of the SOA Order Booking Application"](#)
- [Section 2.4, "Working with the SOA Order Booking Application Code in Oracle JDeveloper"](#)
- [Section 2.5, "Taking a Closer Look at the SOA Order Booking Application"](#)

2.1 Introduction to the SOA Order Booking Application

In this sample application, Global Company sells electronic devices through many channels, including a web-based client application. The ordering process interacts with various internal and external applications to complete the order. These applications include a customer relationship management (CRM) system, a credit validation system, and systems at two external suppliers. For example, the two suppliers, Rapid Manufacturer and Select Manufacturer, are sent information for every order. As part of the order process, they each return a price for which they would supply the items in the order. A condition in the process determines which supplier will be assigned the order.

In order to view the demo (both the code and the applications at runtime), you need to install Oracle SOA Suite and Oracle JDeveloper, then configure the J2EE container for the SOA Order Booking application. You then need to download and deploy the separate applications that are part of the demonstration. Once all are installed and running, you can view the code using Oracle JDeveloper. You can also view the application at runtime by registering as a new customer (or logging in as an existing customer) and placing an order. You can also use the various monitoring tools as the order makes its way through the process.

Read this chapter to understand:

- How to install the SOA Order Booking application
- How to view and work with the SOA Order Booking application
- The separate components and applications that make up the SOA Order Booking application

2.2 Installing the SOA Order Booking Demonstration Application

The SOA Order Booking application runs using Oracle SOA Suite and an Oracle database. To install the SOA Order Booking application, you must:

1. Meet installation requirements and prerequisites. For details, see [Section 2.2.1, "System Requirements"](#) and [Section 2.2.2, "Application Installation Prerequisites"](#).
2. Install the database tables into an Oracle database. For details, see [Section 2.2.3, "Installing the Database Schema"](#).
3. Create a connection pool, data source, and a database adapter connection for the application's container. This allows the application to access the database. For details, see [Section 2.2.4, "Configuring the Container"](#).
4. Deploy the individual applications to the container using Oracle JDeveloper.

2.2.1 System Requirements

There are no system requirements in addition to those to install and run Oracle SOA Suite. For more information, refer to *Oracle Application Server Installation Guide for Microsoft Windows*. You must install Oracle SOA Suite, Oracle JDeveloper, and the SOA Order Booking application on Windows.

2.2.2 Application Installation Prerequisites

The SOA Order Booking application requires an existing Oracle database and Oracle SOA Suite. You deploy the SOA Order Booking application using Oracle JDeveloper.

Do the following before installing the SOA Order Booking application:

- Install an Oracle database: The SOA Order Booking application requires a database for its data. The SQL scripts were written for an Oracle RDBMS, such as 9i, 10g, or XE. The scripts will not install into Oracle Lite. If you wish to use Oracle Lite or some other database, then you will need to modify the database scripts accordingly. You can download an Oracle database from:

<http://www.oracle.com/technology/>

Specifically, Oracle Express Edition (XE) is a small footprint database and is recommended. You can download it from:

<http://www.oracle.com/technology/products/database/xe/>

- Install Oracle SOA Suite. You can download Oracle SOA Suite 10.1.3.1.0 from:

<http://www.oracle.com/technology/soa/>

- Install Oracle JDeveloper: You need Oracle JDeveloper 10.1.3.1 Studio Edition to deploy certain parts of the SOA Order Booking application and to view the application's projects. You can download Oracle JDeveloper from:

<http://www.oracle.com/technology/products/jdev/>

Note: Ensure you download and install 10.1.3.1 (not to be confused with 10.1.3.0) and that it is the Studio edition, not the J2EE or Java edition. You can verify these details in Oracle JDeveloper from the **Help > About** menu option.

- Download the SOA Order Booking application zip file (`SOADEMO.zip`). Extract this zip file to a working directory. This directory will be referred to as *DEMO_HOME* for the remainder of this document. You can download the application zip file from:

<http://www.oracle.com/technology/soa/>

2.2.3 Installing the Database Schema

All the tables required for the SOA Order Booking application are contained in one schema. In a real-world SOA application, it is unlikely that all the tables would be in the same schema, or even the same database or location.

Installing the schema involves creating a user and running SQL scripts that install the schema and populate the tables with sample data.

To install the schema:

1. Connect to your database and create a user named `soademo`.

```
SQL> create user soademo identified by soademo;
SQL> grant resource, connect to soademo;
```

2. Connect to the database as user `soademo`:

```
SQL> conn soademo/soademo
```

3. From the `DEMO_HOME` directory, run the following scripts in the order listed:

- `createSchemaObjects.sql`
- `populateSchemaTables.sql`

Note: Ignore any issues the first time you run these scripts when objects are dropped. These errors are fine, as the objects have not been created yet.

2.2.3.1 The SOA Order Booking Schema

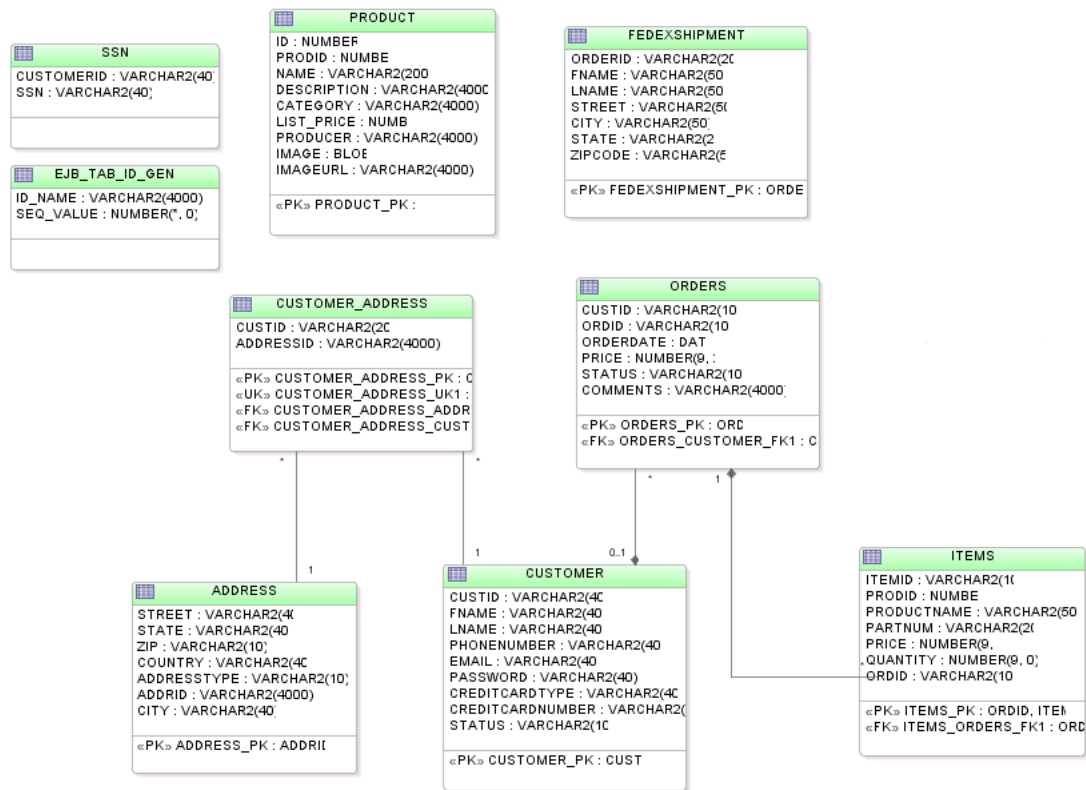
In a real-world SOA application, it is unlikely that all the tables would be in the same schema. [Table 2–1](#) shows each table installed in the `SOADEMO` schema, along with a description of the data stored in the table, the application that uses the table, and where, in a real-world scenario, the table would likely exist.

Table 2–1 Tables in the SOADEMO Schema

Table	Description	Application	Location
ADDRESS	Contains address information.	Customer Service	In the Global Company enterprise, however, it would not exist in the same database as product and order information.
CUSTOMER	Contains customer information, such as name, email, and credit card number.	Customer Service	In the Global Company enterprise, however, it would not exist in the same database as product and order information.
CUSTOMER_ADDRESS	Maps a customer to an address. This mapping table allows a customer to have more than one address, and an address to belong to more than one customer.	Customer Service	In the Global Company enterprise, however, it would not exist in the same database as product and order information.
EJB_TAB_ID_GEN	Sequence table for IDs used in the Customer Service application.	Customer Service	In the Global Company enterprise, however, it would not exist in the same database as product and order information.
FEDEXSHIPMENT	Contains shipping details for an order.	Fulfillment ESB	Federal Express
ITEMS	Contains information regarding the items contained in an order.	SOAOrder Booking	Global Company. This is a "legacy" table that is integrated into the flow, as opposed to a table created for the flow.
ORDERS	Contains information regarding an order. A sequence creates the order IDs.	SOAOrder Booking	Global Company. This is a "legacy" table that is integrated into the flow, as opposed to a table created for the flow.
PRODUCT	Contains the products offered for sale on the web site.	SOADEMO-CLIENT	Global Company

The SSN table is used by the *Oracle Application Server Quick Tour* and is not covered in this document.

Figure 2–1 shows the schema for the SOA Order Booking application.

Figure 2–1 The SOA Order Booking Schema

2.2.4 Configuring the Container

Once you install Oracle SOA Suite, you need to create a connection pool, data source, and database adapter for the SOA Order Booking application.

To configure the container:

1. Point your browser to the Enterprise Manager for your installation of Oracle SOA Suite. You will use the Enterprise Manager to configure the application server and containers.

For example:

`http://localhost:8888`

8888 is the default HTTP port. Use the port number that the installer selected for your environment. If port 8888 does not work for you and you are not sure what your port number is, use a text editor and open this file:

`ORACLE_HOME\install\readme.txt`

Look for the line Access the Oracle SOA Suite 10g Application Server Welcome Page via the following URL and the next line will show you the URL for your Enterprise Manager welcome page.

2. From the right-hand side **Manage Your SOA Suite** portlet, click the **Application Server Control** link.

The Application Server Control Console displays the administrator logon dialog box.

3. Enter a valid username and password for Oracle Enterprise Manager 10g. By default, `oc4jadmin` is the administrator user. The password for this user was set during the Oracle SOA Suite installation.

4. Create a connection pool and JDBC data source for the OC4J instance.

A data source enables you to retrieve a connection to a database server. A managed data source uses a connection pool to efficiently manage a database connection. You must define one connection pool and its connection factory.

- a. Select the **home** OC4J link.

The OC4J: home page appears.

- b. Click the **Administration** tab.

- c. In **Services > JDBC Resources**, click the **Go To Task** icon.

The JDBC Resources pages appears.

- d. In the **Connection Pools** section, click the **Create** button.

The Create Connection Pool - Application page appears.

- e. Accept defaults, and click **Continue**.

The Create Connection Pool page appears.

- f. Enter the following values, leaving the defaults for the other fields:

Element	Value
Name	soademo_pool
JDBC URL	URL for your database. For example: jdbc:oracle:thin:@localhost:1521:orcl
Username	soademo
Password	Password entered when creating the soademo user in the database

- g. Click **Test Connection**.

The Test Connection page appears.

- h. Click **Test**.

The JDBC Resources page updates with a successful connection confirmation message. If you get an error message, check the URL and credentials to ensure that you entered the right information.

- i. Click **Finish**.

- j. Under **Data Sources**, click **Create**.

The Create Data Source - Application & Type page appears.

- k. Accept the defaults, and click **Continue**.

The Create Data Source - Managed Data Source page appears.

- l. Enter the following values, leaving the defaults for the other fields:

Element	Value
Name	soademoDS
JNDI Location	jdbc/soademoDS
Connection Pool	soademo_pool

- m. Click **Finish**.

- n. Back in the **Data Sources** section, locate the soademoDS data source, and click the **Test Connection** icon.

The Test Connection page appears.

- o. Click **Test**.

The JDBC Resources page updates with a successful connection confirmation message.

5. Create a database adapter connection factory.

This connection factory will be used by an outbound database adapter within a process flow written in Business Process Execution Language (BPEL), using the Java Transaction API (JTA). Because of this, you need to define an `xaDataSourceName` that points to a transaction-level global data source in the `data-sources.xml` file, as opposed to the locally transacted data source (`dataSourceName`).

- a. At the top of the JDBC Resources page, click the **OC4J:home** breadcrumb link.

The OC4J: home page appears.

- b. Click the **Applications** tab.

- c. In the table of applications, click the **default** link.

The Application: default page appears.

- d. In the **Modules** section, click the **DbAdapter** link.

The Resource Adapter: Database Adapter page appears.

- e. Click the **Connection Factories** tab.

- f. At the top of the page, click the **Create** button.

Note: Do not click the **Create** button in the **Shared Connection Pools** section.

The Create Connection Factory: Select Interface page appears.

- g. Accept the default interface, and click **Continue**.

The Create Connection Factory page appears.

- h. In the **JNDI Location** field, enter `eis/DB/soademo`.

- i. In the **Configuration Properties** section, for **xaDataSourceName**, enter `jdbc/soademoDS`.

- j. Leave the defaults for the other fields.

- k. Click **Finish**.
- l. Leave the Application Server Control Console running, as you need it for the next task.

The container is now configured for the SOA Order Booking application.

2.2.5 Noting the Important Port Numbers

The SOA Order Booking application has some configuration files that use the default port numbers. Your installation may not be using the default port numbers and may need to be reconfigured. you can use the Application Server Control Console to find the port numbers being used in your environment.

To find the port numbers:

1. From Application Server Control Console, click the **Cluster Topology** breadcrumb link.
2. In the **Administration** section towards the bottom of the page, select **Runtime Ports**.

The Runtime Ports page shows the ports being used by your installation, as shown in [Figure 2–2](#).

Figure 2–2 The Runtime Ports Page

ORACLE Enterprise Manager 10g
Application Server Control

Setup Logs Help Logout

Cluster Topology >
Runtime Ports

Expand All | Collapse All

Name	Host	Port In Use	Port Range	Configure Port
▼ All Application Servers				
▼ soademo.dsteiner-pc.us.oracle.com	dsteiner-pc			
▼ OPMN				
Local		6100	6100	
Remote		6200	6200	
Request		6003	6003	
▼ OC4J : home				
JMS		12601	12601-12700	
HTTP		8888		
RMIS		12701	12701-12800	
RMI		12401	12401-12500	

Setup | Logs | Help | Logout

Copyright © 1996, 2006, Oracle. All rights reserved.

These are the important ones to note for the SOA Order Booking application installation:

- OPMN Request (defaults to 6003)
- OC4J:home HTTP (defaults to 8888)
- OC4J:home RMI (defaults to 12401)

If your ports are the same as the defaults, listed above, then you will not need to change any configuration files used in deployment, as they already use those defaults. If one or more of your ports are different, then note them down for now. In some later installation steps you will be asked to refer to these port numbers.

2.2.6 Creating Connections in Oracle JDeveloper

Before you can use Oracle JDeveloper to deploy the application to the server, Oracle JDeveloper needs to know how to connect to the server. This includes the application server and the database.

To create these connections in Oracle JDeveloper:

1. Start Oracle JDeveloper by running `JDEV_HOME\jdeveloper.exe`.
2. Create a connection to the database using the `SOADEMO` schema:
 - a. Click the **Connections** tab, or if it not currently displayed, choose **View > Connection Navigator**.
 - b. Double-click on the **Database** folder to open the Create Database Connection wizard.
 - c. Complete the wizard, ensuring that you enter the following:
 - On the Step 1 of 4: Type 1 page, enter `soademo` for **Connection Name**.
 - Enter the appropriate values for where you installed the schema.
3. Create the application server connection:
 - a. Double-click **Application Server** to open the Create Application Server Connection wizard.
 - b. On the Step 1 of 4: Type page, perform the following and then click **Next**.

Element	Action
Connection Name	Enter a connection name of your choice.
Connection Type	Select Oracle Application Server 10g 10.1.3 .

- c. On the Step 2 of 4: Authentication page, perform the following and then click **Next**.

Element	Action
User	Enter a valid user name for the application server.
Password	Enter a valid password for the user name.
Deploy Password	Click this checkbox.

- d. On the Step 3 of 4: Connection page, perform the following and then click **Next**.

Element	Action
Host Name	Enter the host name in which you installed Oracle SOA Suite.
OPMN	Enter the OPMN request port. The default is 6003. See Also: Section 2.2.5, "Noting the Important Port Numbers" to find out what the port number is for your environment
Group/OC4J Instance Name	Enter home.

- e. On the Step 4 of 4: Test page, click **Test the Connection**.

If the test does not succeed, use the **Back** button to verify and change values.

- f. Click **Finish**.

4. Create a connection to the Oracle BPEL Process Manager and Oracle Enterprise Service Bus (ESB) integration server.

Creating this connection enables you to deploy business processes to Oracle BPEL Process Manager and services to the Enterprise Service Bus.

- Double-click **Integration Server** to open the Create Integration Server Connection wizard.
- On the Step 1 of 3: Name page, enter a name of your choice.
- On the Step 2 of 3: Connection page, perform the following and then click **Next**.

Element	Action
Application Server	Select the name of the application server you created in Step 3.
Host Name	Enter the host name in which you installed Oracle SOA Suite
Port	Enter the port for the Oracle HTTP Server installed with Oracle SOA Suite. The default is 8888. See Also: Section 2.2.5, "Noting the Important Port Numbers" to find out what the port number is for your environment

- d. On the Step 3 of 3: Test page, click **Test the Connection**.

If the test does not succeed, use the **Back** button to verify and change values.

- e. Click **Finish**.

2.2.7 Deploying the Separate SOA Order Booking Applications

The SOA Order Booking application actually consists of a number of applications orchestrated to work together to fulfill a customer's order. Specifically, the following applications are part of the SOA Order Booking application:

- **CreditService:** Validates a customer's credit information.
- **CustomerService:** Represents an existing CRM application within the Global Company enterprise.

- **FulfillmentESB:** Provides the ESB flow used to determine the shipping vendor for an order
- **OrderBookingESB:** Provides the ESB flow used to route order information from the web client and invoke the Order Booking BPEL flow. While the web client could directly invoke the BPEL flow, ESB is used so that more than one client can submit an order.
- **RapidService:** Provides the web service used to communicate with Rapid Manufacturer to provide a price for a given order. This service is meant to demonstrate a synchronous service interaction.
- **SelectManufacturer:** Provides the BPEL flow used by Select Manufacturer to provide a price for a given order. This flow is meant to demonstrate integrating with an asynchronous service.
- **SOADEMO-CLIENT:** Provides the web-based application that enables end users to register themselves as customers, and then order electronics online from Global Company.
- **SOAOrderBooking:** Provides the BPEL flow that handles the processing of the order.

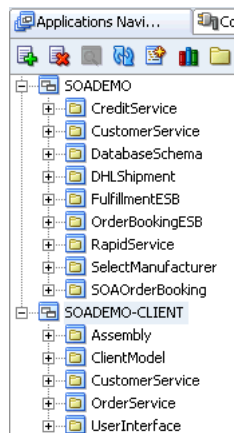
Note: The SOA Order Booking application also contains two other projects. The `DatabaseSchema` project contains the scripts needed to create the schema for the applications. The `DHLShipment` project is used only by *Oracle Application Server Quick Tour*, and is not documented in this guide.

You use the connections you created in Oracle JDeveloper to deploy the different applications included in the SOA Order Booking application.

To deploy the SOA Order Booking application:

1. Open the applications in Oracle JDeveloper:
 - a. In the JDeveloper main menu, choose **File > Open**.
 - b. Navigate to `DEMO_HOME` and open the `SOADEMO.jws` file.
 - c. Choose **File > Open** again, and navigate to `DEMO_HOME/SOADEMO-CLIENT` and open the `SOADEMO-CLIENT.jws` file.

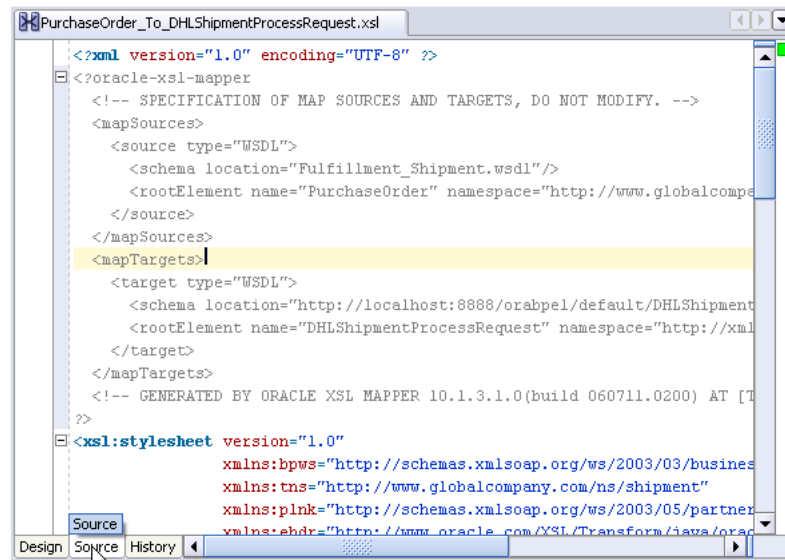
Figure 2–3 shows the Application Navigator after you open the two application files. For a description of each of these applications, see [Section 2.3, "An Overview of the SOA Order Booking Application"](#).

Figure 2–3 The SOA Order Booking Applications in Oracle JDeveloper

Caution: The following steps MUST be executed in the order documented, or the applications will not properly deploy.

2. Deploy the FulfillmentESB project:
 - a. Expand SOADEMO.
 - b. If you are using port 8888 as your HTTP port, then proceed to Step g. If you are not using the default HTTP port, expand the **Resources** node and double-click the `PurchaseOrder_To_DHLShipmentProcessRequest.xml` file to open it.

Disregard any errors that appear.
 - c. In the `PurchaseOrder_To_DHLShipmentProcessRequest.xml` file, replace all instances of 8888 with your port number. See [Section 2.2.5, "Noting the Important Port Numbers"](#) to find out what the port number is for your environment.
 - d. Click the Source sub-tab, located at the bottom of the `PurchaseOrder_To_DHLShipmentProcessRequest.xml` page, to switch to the Source view, as shown in [Figure 2–4](#).

Figure 2–4 Source Tab

- e. Replace all instances of 8888 with your port number. See [Section 2.2.5, "Noting the Important Port Numbers"](#) to find out what the port number is for your environment.
 - f. Choose **File > Save** to save your work.
 - g. Right-click FulfillmentESB and select **Register with ESB > <IntegrationServer_Connection_Name>**.
 - h. Click **OK** in the ESB Registration Summary dialog.
3. Deploy the Select Manufacturer BPEL process flow using Ant:
 - a. Expand **SelectManufacturer/Resources**.
 - b. Open the build.properties file and verify that the values are correct.
If the values are not correct, change the values and then save the file. To determine correct port information, see [Section 2.2.5, "Noting the Important Port Numbers"](#).
 - c. In the Application Navigator, right-click build.xml and choose **Run Ant**.
 - d. Configure Ant to use the build.properties file:
 - * Click the **Properties** tab.
 - * Remove all properties.
 - * In the **Property Files** section, click **Add**.
 - * In the Add Ant Property dialog, select the build.properties file, click **Open**, and then click **OK**.

This will start the deployment process, which may take anywhere from 30 to 60 seconds. You can monitor the progress in the Messages window, in the Apache Ant - Log tab, in the Apache Ant subtab. You will know the deployment is complete when you see the text BUILD SUCCESSFUL.

4. Deploy the Credit Service application:
 - a. Expand **CreditService/Resources**.
 - b. Right-click the `WebServices.deploy` file, and choose **Deploy to > <ApplicationServer_Connection_Name>**.
 - c. In the Configure Application dialog, click **OK**.

When deployment completes, the Deployment - Log displays a Deployment finished message.
5. Deploy the Customer Service application:
 - a. Expand **CustomerService/Resources**.
 - b. Right-click the `CustomerService.deploy` file and choose **Deploy to > <ApplicationServer_Connection_Name>**.
 - c. Click **OK** in the Configure Application dialog.

When deployment completes, the Deployment - Log displays a Deployment finished message.
6. Deploy the Rapid Service application:
 - a. Expand **RapidService/Resources**.
 - b. Right-click the `WebServices.deploy` file and choose **Deploy to > <ApplicationServer_Connection_Name>**.
 - c. Click **OK** in the Configure Application dialog.

When deployment completes, the Deployment - Log displays a Deployment finished message.
7. Deploy the SOA Order Booking BPEL flow using Ant:
 - a. Expand **SOAOrderBooking/Resources**.
 - b. Open the `build.properties` file, and verify that the values are correct.

If the values are not correct, change the values, and then save the file. To determine correct port information, see [Section 2.2.5, "Noting the Important Port Numbers"](#).
 - c. If you are using port 8888 as your HTTP port, then proceed to Step k. If you are not using the default HTTP port, expand the Integration Content node and double-click the `bpel.xml` file to open it.
 - d. In the `bpel.xml` file, replace all instances of 8888 with your port number. See [Section 2.2.5, "Noting the Important Port Numbers"](#) to find out what the port number is for your environment.
 - e. Choose **File > Save** to save your work, and then restart Oracle JDeveloper.
 - f. Under the **Integration Content** node, double-click the `CreditValidatingService.wsdl` file to open it.
 - g. Click the Source sub-tab to switch to the Source view.
 - h. Replace all instances of 8888 with your port number. See [Section 2.2.5, "Noting the Important Port Numbers"](#) to find out what the port number is for your environment.
 - i. Choose **File > Save** to save your work.
 - j. Repeat Steps f through i for `CustomerSvc.wsdl` and `RapidService.wsdl`.

- k. Right-click the `build.xml` file and choose **Run Ant**.
- l. Configure Ant to use the `build.properties` file.
 - * Click on the Properties tab.
 - * Remove any properties.
 - * In the **Property Files** section, click **Add**.
 - * In the Add Ant Property File, select the `build.properties` file, click **Open**, and then click **OK**.

This will start the deployment process, which may take anywhere from 30 to 60 seconds. You can monitor the progress in the Messages window, in the Apache Ant - Log tab, in the Apache Ant subtab. You will know the deployment is complete when you see the text `BUILD SUCCESSFUL`.

- 8. Deploy the Order Booking ESB project:
 - a. If you are using port 8888 as your HTTP port, then skip this step and proceed to Step c. Using a text editor, open the following files and change all references of 8888 with your port number. See [Section 2.2.5, "Noting the Important Port Numbers"](#) to find out the HTTP port for your environment.

```
DEMO_HOME\OrderBookingESB\OrderBooking.esbsys
```

```
DEMO_HOME\OrderBookingESB\OrderBooking_
OrderBookingProcess.esbsvc
```

- b. Choose **File > Save** to save your work.
 - c. Right-click **OrderBookingESB** and choose **Register with ESB > <IntegrationServer_Connection_Name>**.
 - d. Click **OK** in the Summary dialog.

- 9. Deploy the SOADEMO-CLIENT web client:

- a. If you are using port 8888 as your HTTP port, then skip this step and proceed to Step c. Using a text editor, open the following files and change all references of 8888 with your port number. See [Section 2.2.5, "Noting the Important Port Numbers"](#) to find out the HTTP port for your environment.

```
DEMO_
HOME\SOADEMO-CLIENT\CustomerService\src\oracle\soademo\vie
w\services\runtime\CustomerServiceSoapHttp_Stub.java
```

```
DEMO_
HOME\SOADEMO-CLIENT\OrderService\src\oracle\soademo\view\s
ervices\runtime\__soap_initiate_ppt_Stub.java
```

- b. Choose **File > Save** to save your work.
 - c. Expand **SOADEMO-CLIENT/Assembly/Application Sources**.
 - d. Right-click the `SOADEMO.deploy` file and choose **Deploy to > <ApplicationServer_Connection_Name>**.
 - e. Click **OK** in the Configure dialog.

2.2.8 Configuring the ESB Port

If your HTTP port is 8888, which is the default, then you do not need to configure the ESB port. Otherwise, follow these steps:

To configure the ESB port:

1. Point your browser to the Welcome to Oracle SOA Suite (10.1.3.1.0) page. For example:

`http://localhost:8888`

Tip: To determine the port number is for your environment, see [Section 2.2.5, "Noting the Important Port Numbers"](#).

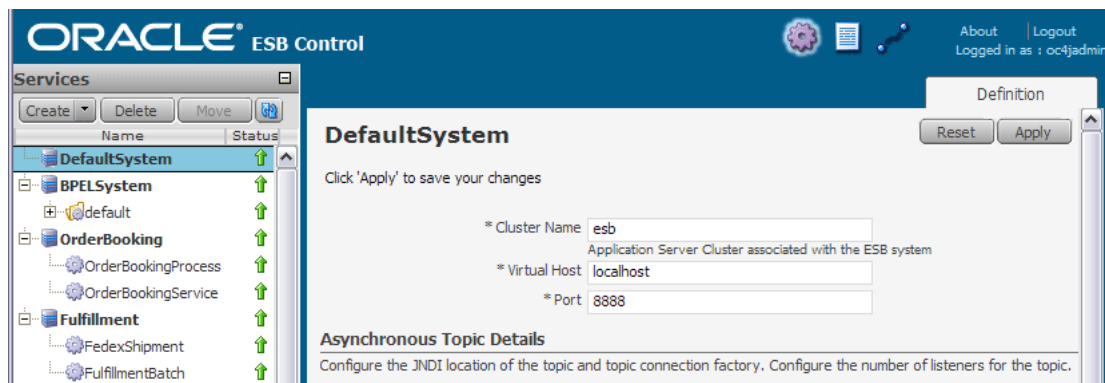
2. From the right-hand side **Manage Your SOA Suite** portlet, click the **ESB Control** link.

The Oracle Enterprise Manager 10g ESB Control displays.

3. In the **Services** pane, select the **DefaultSystem** group.

Configuration information appears in the right-hand pane.

Figure 2–5 Oracle ESB Control



4. In the **Port** field, change the HTTP port to the HTTP port number used in your environment.

Tip: To determine the port number is for your environment, see [Section 2.2.5, "Noting the Important Port Numbers"](#)

5. Click **Apply**.
6. Repeat Steps 3 through 5 for other groups in the **Services** pane: **BPELSystem**, **OrderBooking**, and **Fulfillment**.
7. Close the Oracle Enterprise Manager 10g ESB Control.

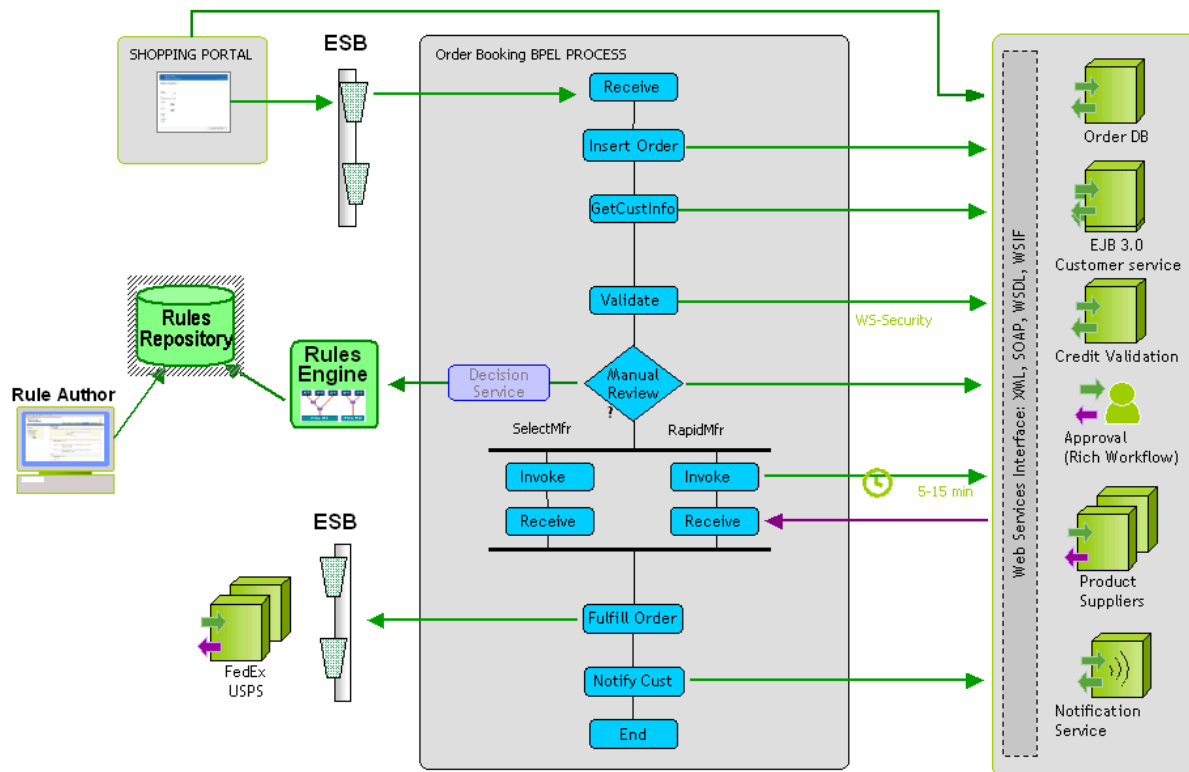
2.3 An Overview of the SOA Order Booking Application

Global Company sells electronic devices through many channels, including the web. Customers can visit the web site, register, and place orders for the products. In order to register customers and fulfill orders, several applications both internal and external to Global Company work together in the following way:

- When a new customer registers in the web client, a web service invokes the CRM application, which then stores the customer information in its database.
- When a registered customer attempts to log into the web client, the web client again uses a web service to invoke the CRM application, which provides authentication.
- When a customer places an order, the following takes place:
 1. A flow in Oracle ESB receives a message containing order information, transforms the message for a BPEL process flow, and invokes that process, providing it with the transformed order information.
 2. A BPEL flow then:
 - Sets the order to a pending state, and inserts the order information into the database.
 - Retrieves customer information by placing a synchronous call to a web service that retrieves information from the CRM application.
 - Validates the retrieved credit card information through a synchronous call to the external credit validation service. If the credit card is not valid, the process cancels the order. If the card is approved, the flow continues.
 - Uses a decision service to determine whether or not the order requires manual approval. The decision service uses a rule that states the following: If the customer has platinum status, the order does not require approval. If the customer has gold status, approval is required only for orders over \$1,000. If the customer has silver status, then all orders must be approved.
 - If the order requires manual approval, a human workflow routes a message to an employee at Global Company who has authority to approve orders. The employee then uses the Oracle BPEL Worklist Application web-based application to view the order, and either approve or disapprove it. The message is sent back into the BPEL flow. If the order is not approved, the process cancels the order. If it is approved, the flow continues.
 - Once the order is approved, or if the order does not require manual approval, sends the order information to two suppliers: Rapid Manufacturer and Select Manufacturer. Each supplier returns a price (one synchronously and one asynchronously). The flow then uses a conditional expression that determines which supplier returned the lower price, and assigns the order to that supplier.
 - The flow then uses Oracle ESB routing rules to determine the vendor that will ship the order. If the order is over \$500, the order information is sent to Federal Express. If it is under \$500, the information is sent to the United States Postal Service.
 - Sets the order to complete, and invokes a notification service, which sends an email to the customer.

Figure 2–6 shows the flow of information through the various components and applications that make up the SOA Order Booking application.

Figure 2–6 The SOA Order Booking Application Flow



For a detailed description of each how each application works at runtime, see [Section 2.5, "Taking a Closer Look at the SOA Order Booking Application"](#).

2.4 Working with the SOA Order Booking Application Code in Oracle JDeveloper

Once you have opened the applications in Oracle JDeveloper, you can then begin to review the artifacts within each application. The development environment for the SOA Order Booking application is divided into two applications: the SOADEMO application and the SOADEMO-CLIENT application.

Note: For the purposes of the demo, all code is delivered in a preconfigured development environment. In a real-world development environment, you would not have access to much of this code. Instead, you would be given only WSDL files for the different applications with which you need to integrate. Additionally, in many cases, the code for the applications is not indicative of a real-world application, and contains only the functionality needed to demonstrate integrating within an SOA framework.

For example, the code for the customer service application is very rudimentary, and exists as a project in the same application as the credit service application. In actuality, the credit service application would exist at a company external to Global Company.

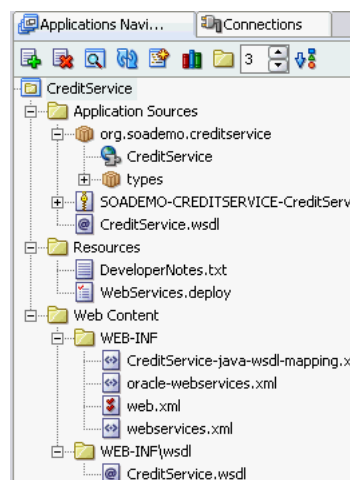
2.4.1 The SOADemo Application

The SOADEMO application contains separate projects, which in turn contain the code for the different applications used in the SOA Order Booking demonstration application.

2.4.1.1 The CreditService Project

The credit service application resides in the `CreditService` project. [Figure 2–7](#) shows the `CreditService` project and its associated directories:

Figure 2–7 The CreditService Project



This project contains the following directories:

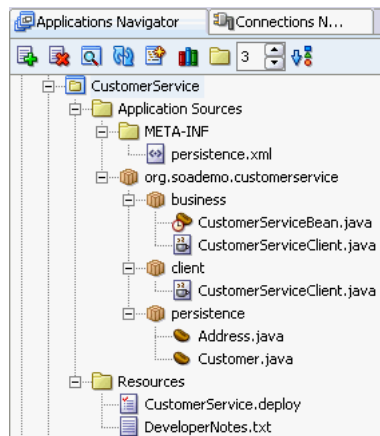
- **Application Sources:** Contains the CreditService web service generated from a provided wsdl file, the actual CreditService.wsdl file, along with an EAR file that contains the actual CreditService application that is deployed to the application server.
- **Resources:** Contains the deployment configuration file.
- **Web Content:** Contains the files needed to deploy a web service. These are automatically generated when creating web services from WSDL files in Oracle JDeveloper.

For information about creating web services from provided WSDL files, see [Section 5.2.6, "How to Create a Top-Down Web Service"](#). For information about how the CreditService project is used at runtime, see [Section 2.5.3.3.3, "CreditService Scope"](#).

2.4.1.2 The CustomerService Project

The customer service application resides in the CustomerService project. [Figure 2–8](#) shows the CustomerService project and its associated directories.

Figure 2–8 The Customer Service Project



This project contains the following directories:

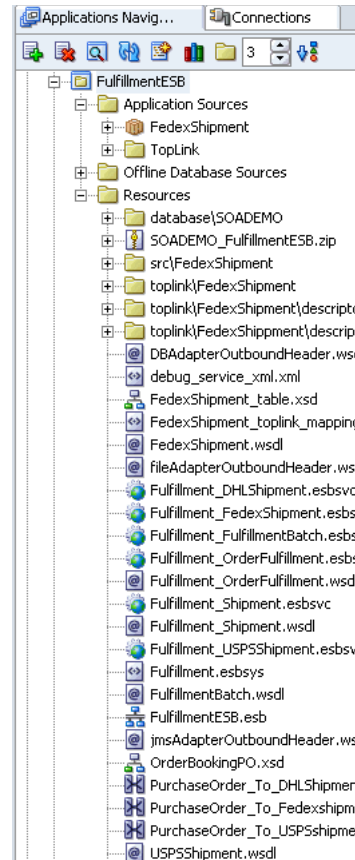
- **Application Sources:** Contains the source for the customer service application.
- **Resources:** Contains the deployment configuration file.

For information about using Enterprise JavaBeans (EJB) to create a service, see [Chapter 4, "Building and Using Application Services"](#). For information about how the CustomerService project is used at runtime, see [Section 2.5.1, "Registration Process"](#), [Section 2.5.2, "The Login Process"](#), and [Section 2.5.3.3.2, "CustomerService Scope"](#).

2.4.1.3 The FulfillmentESB Project

The files required for the fulfillment ESB flow reside in the `FulfillmentESB` project. [Figure 2–9](#) shows the `FulfillmentESB` project and its associated directories.

Figure 2–9 The FulfillmentESB Project



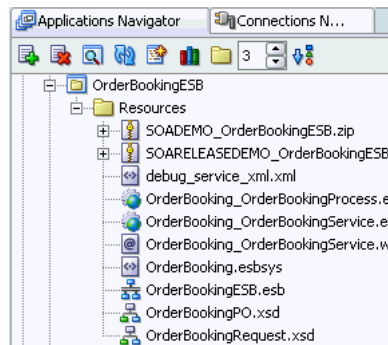
This project contains the following directories:

- **Application Sources:** Contains the source for the code for the `FedexShipment` application that populates the `FEDEXSHIPMENT` table.
- **Offline Database Sources:** Representations of the tables used by the database adapter.
- **Resources:** Contains the files for the ESB flow, including schemas and WSDL files.

For information about using ESB, see [Chapter 6, "Using Oracle Enterprise Service Bus"](#). For information about how the `FulfillmentESB` project works at runtime, see [Section 2.5.3.4, "OrderFulfillmentESB Flow"](#).

2.4.1.4 The OrderBookingESB Project

The files required for the OrderBooking ESB flow reside in the `OrderBookingESB` project. [Figure 2–10](#) shows the `OrderBookingESB` project and its associated directories.

Figure 2–10 The OrderBookingESB Project

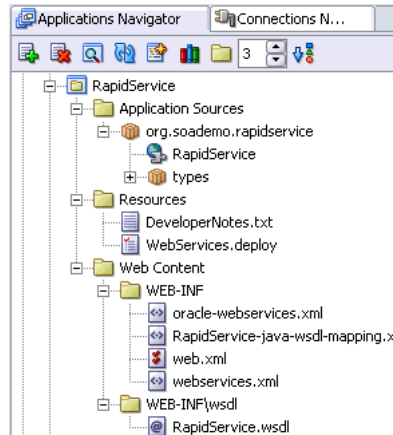
This project contains the following directories:

- **Resources:** Contains all the files for the ESB flow.

For more information about using ESB, see [Chapter 6, "Using Oracle Enterprise Service Bus"](#). For information about how the OrderBookingESB project works at runtime, see [Chapter 2.5.3.2, "The OrderBookingESB Flow"](#).

2.4.1.5 The RapidService Project

The application that allows Rapid Manufacturer to provide prices back to the flow resides in the RapidService project. [Figure 2–11](#) shows the RapidService project and its associated directories.

Figure 2–11 The RapidService Project

This project contains the following directories:

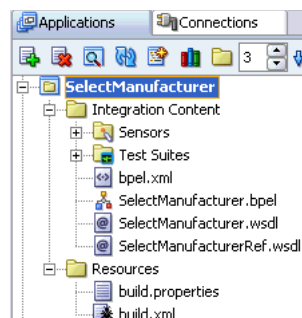
- **Application Sources:** Contains the web service to Rapid Manufacturer.
- **Resources:** Contains the deployment configuration file.
- **Web Content:** Contains the files needed to deploy a web service. These are automatically generated when creating web services in Oracle JDeveloper.

For information about creating a web service from a Java class, see [Section 5.2.3, "How to Create a Bottom-Up Web Services"](#). For information about how the RapidService project works at runtime, see [Section 2.5.3.3.6, "SelectSupplier Scope"](#).

2.4.1.6 The SelectManufacturer Project

The files required for the SelectManufacturer BPEL flow reside in the SelectManufacturer project. [Figure 2–12](#) shows the SelectManufacturer project and its associated directories.

Figure 2–12 The SelectManufacturer Project



This project contains the following directories:

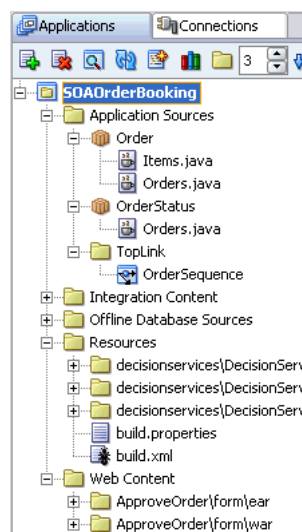
- Integration Content: Contains the files for the BPEL flow, including sensors, test suites, and WSDL files.
- Resources: Contains the deployment files.

For more information about BPEL flows, see [Chapter 7, "Using Oracle BPEL Process Manager"](#). For information about how the SelectManufacturer project works at runtime, see [Chapter 2.5.3.3.6, "SelectSupplier Scope"](#).

2.4.1.7 The SOAOrderBooking Project

The files required for the OrderBooking BPEL flow reside in the SOAOrderBooking project. [Figure 2–13](#) shows the SOAOrderBooking project and its associated directories.

Figure 2–13 The SOAOrderBooking Project



This project contains the following directories:

- **Application Sources:** Contains the source for the application that populates the ORDERS and ITEMS tables, and updates the status.
- **Integration Content:** Contains the files for the BPEL flow, including sensors, test suites, and WSDL files.
- **Offline Database Sources:** Representations of the tables used by the database adapter.
- **Resources:** Contains the deployment files and directories for the decision service, which is used to invoke business rules.
- **Web Content:** Contains the files for the manual approval web page.

For more information about BPEL flows, see [Chapter 7, "Using Oracle BPEL Process Manager"](#). For information about how the SOAOrderBooking project works at runtime, see [Section 2.5.3.3, "SOAOrderBooking BPEL Flow"](#).

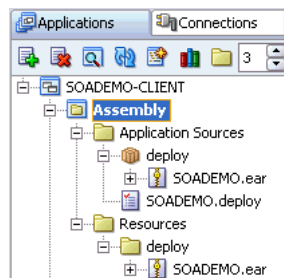
2.4.2 The SOADemo-Client Application

The SOADEMO-CLIENT application contains projects for the web client application. The files required for the web client reside in the SOADEMO-CLIENT application.

2.4.2.1 The Assembly Project

The Assembly project contains all the files necessary to deploy the web application. [Figure 2–14](#) shows the Assembly project and its associated directories.

Figure 2–14 The Assembly Project



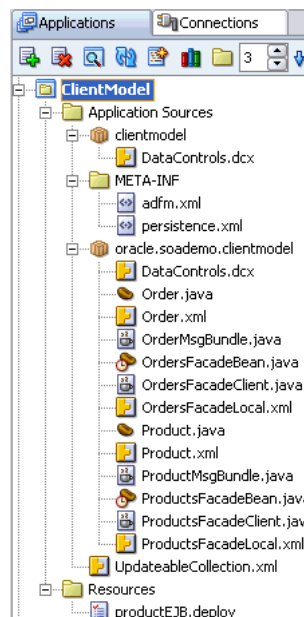
This project contains the following directories:

- **Application Sources:** Contains the files necessary for deployment.
- **Resources:** Contains the EAR file.

For information about deploying web applications, see [Chapter 11, "Deploying SOA Applications with Enterprise Manager"](#).

2.4.2.2 The ClientModel Project

The ClientModel Project contains the classes that allow the product data to be displayed in the web application. [Figure 2–15](#) shows the ClientModel project and its associated directories.

Figure 2–15 The ClientModel Project

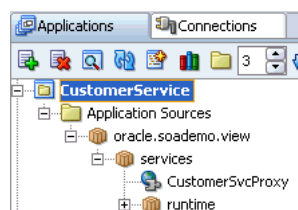
This project contains the following directories:

- **Application Sources:** Contains the files used to access the product data. Included are the metadata files used by Oracle Application Development Framework (ADF) to bind the data to the view.
- **Resources:** Contains a file used in deployment.

For information about creating classes that use the Java Persistence API to manage data, see [Section 4.2, "Creating JPA Entities that Map to Database Tables"](#). For information about using ADF in web applications, see [Chapter 9, "Developing a Web Application in an SOA System"](#).

2.4.2.3 The CustomerService Project

The CustomerService project contains the proxy classes for the customer service application. [Figure 2–16](#) shows the CustomerService project and its associated directory.

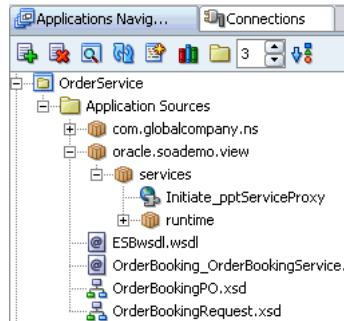
Figure 2–16 The CustomerService Project

The project contains the **Application Sources** directory, which holds the proxy to the customer service application. For information about creating web service proxies, see [Section 5.3.1, "How to Create a Web Service Proxy"](#).

2.4.2.4 The OrderService Project

The `OrderService` project contains the proxy classes used to send order information into ESB. [Figure 2–17](#) shows the `OrderService` project and its associated directory.

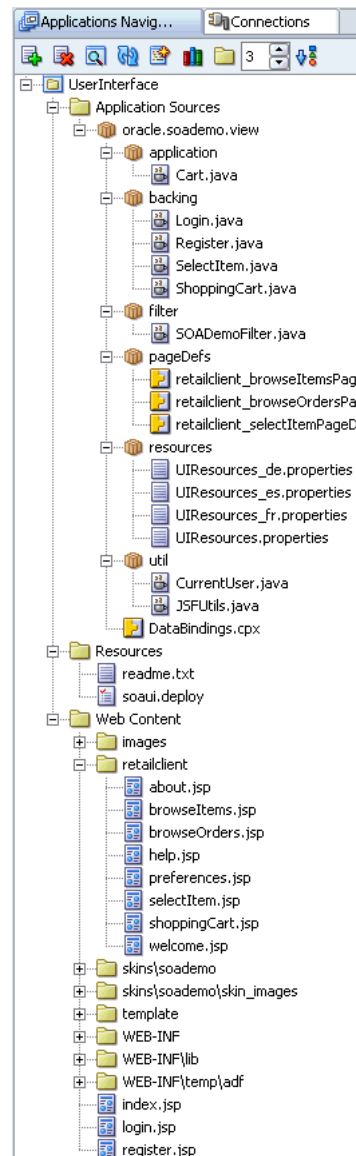
Figure 2–17 The OrderService Project



The project contains the `Application Sources` directory, which holds the proxy to the web service that invokes the `OrderBookingESB`, along with the files generated when you create the proxy. For information about creating web service proxies, see [Section 5.3.1, "How to Create a Web Service Proxy"](#).

2.4.2.5 The UserInterface Project

The `UserInterface` project contains the files for the web interface, including the backing beans, deployment files, and JSPX files. [Figure 2–18](#) shows the `UserInterface` project and its associated directories.

Figure 2–18 The UserInterface Project

This project contains the following directories:

- **Application Sources:** Contains the code used by the web client, including the managed and backing beans, property files used for internationalization, and the metadata used by ADF to display bound data.
- **Resources:** Contains the files used to deploy the application.
- **Web Content:** Contains the web files, including the JSP files, images, skin files, deployment descriptors, and libraries.

For more information about creating web clients, see [Chapter 9, "Developing a Web Application in an SOA System"](#).

2.5 Taking a Closer Look at the SOA Order Booking Application

You run the SOA Order Booking application by running the SOADEMO-CLIENT application. This application uses JavaServer Faces (JSF) as the view technology, and relies on backing beans to invoke the web services for the other applications.

To launch the client, open a browser and enter the following URL:

```
http://<host_name>:<port_number>/soademo
```

For example:

```
http://localhost:8888/soademo
```

Once the login page launches, you can use a valid email/password combination to enter the web site, or you can choose to register as a new customer. The SOA Order Booking application ships with predefined customer data. You can use one of these customers' email and password to log in, or you can register as a new customer.

[Table 2–2](#) shows the preregistered customers.

Table 2–2 Preregistered Customers in the SOA Order Booking Application

Email	Password	Customer Status	Notes
sking@soademo.org	welcome1	Gold	Requires manual approval for orders over \$1,000.
jchen@soademo.org	welcome1	Platinum	Has an invalid credit card number.
ghimuro@soademo.org	welcome1	Silver	Requires manual approval for all orders.

Creating an order invokes the ESB that then invokes the BPEL process flow that handles the order. Once the order is in-flight, you can use the ESB console to view the status of the ESB, and the BPEL console to view the status of the process flow. You can also use Oracle Enterprise Manager to view the status of the entire system.

While the SOA Order Booking application does not by default have any security constraints placed on the services, you can secure the services using Oracle Web Services Manager (Oracle WSM). This application contains a tool to define security for your services, as well as a console that allows you to monitor your services. For more information about using Oracle WSM, see [Chapter 10, "Securing the System"](#).

2.5.1 Registration Process

When a user registers as a customer, the web client collects the needed information, and then sends that information to the Customer Service application by invoking the Customer Web service. [Figure 2–19](#) shows the Registration page.

Figure 2–19 Registration Page

The screenshot shows a web form titled "Application Registration". It contains several sections of input fields:

- Personal Information:** Fields for * First Name, * Last Name, * Phone, Street, City, State, and Zip Code.
- Login Credentials:** Fields for * Email, * Password, and * Re-enter.
- Credit Card Information:** Fields for * Credit Card Number and a dropdown for Credit Card Type.

At the bottom of the form are two buttons: "Register" and "Cancel".

When the user clicks the **Register** button, the data is collected by a method on the `Register.java` backing bean, which then invokes the `CustomerService` web service. The CRM application then persists the data to the `Customer` tables.

Where to Find Implementation Details

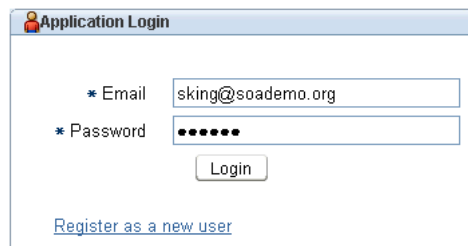
Following are the sections of *Oracle SOA Suite Developer's Guide* that describe how to develop the components used in the registration process:

- **Collecting information on a web page:** The registration page collects information by binding JSF input components to properties on a backing bean. The page was designed to collect the needed information as described by the `CustomerService` web service. For information about creating web pages that collect information needed by web services, see [Section 9.6, "Collecting Data for Use by a Web Service"](#).
- **Invoking a web service from a web page:** The `register.java` JSF backing bean contains a method that invokes a web service and populates the web service with the registration data. For more information, see [Section 9.6.3, "How to Invoke a Service Using a Backing Bean"](#).

2.5.2 The Login Process

When a registered user logs into the web client, the user name and password are compared with the credential information stored in the customer service database. [Figure 2–20](#) shows the Application Login page for the web client.

Figure 2–20 Application Login Page



When the user clicks the **Login** button, the associated `Login.java` backing bean collects that information and invokes the `CustomerService` web service, which then verifies the user information in the database.

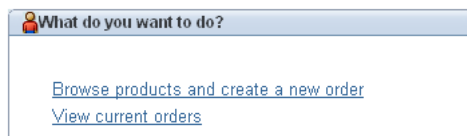
2.5.3 The Ordering Process

The ordering process begins in the web client application where a user can shop for and order products. The web application kicks off an ESB flow, which in turn invokes a BPEL flow. The BPEL flow handles the actual ordering process.

2.5.3.1 The Web Client

Registered users can shop for products by clicking the **Browse products** link on the Welcome page, which displays directly after users log in, as shown in [Figure 2–21](#).

Figure 2–21 Application Welcome Page



This link displays the Browse Items page, which lists all products available for sale, as shown in [Figure 2–22](#). The product information is stored in the `PRODUCT` table, in the client application's database. That data is displayed in a table that uses ADF data binding.

Figure 2–22 Booking Browse Items Page

Select and		View Details		Previous 1-10 of 27 Next 10		
Select	Name	Product Id	Description	Category	Price	Producer
☒	HD Television	128723	A High-Definition Flat Screen Television that uses DLP Technology.	Video	\$1999.00	Azimuth
☐	PlayStation 2 Video Game System	223453	Playstation II Interactive Video Gaming System.	Games	\$199.00	Playstation
☐	Treo 650 Phone/PDA	429832	A fully featured Treo Phone and PDA combination.	Handhelds	\$299.00	Treo
☐	Treo 700w Phone/PDA	423322	A fully featured Treo Phone and Windows CE Handheld.	Handhelds	\$399.00	Treo
☐	Tungsten E PDA	592039	Tungsten handheld Personal Digital Assistant.	Handhelds	\$195.00	Tungsten
☐	XBox Video Game System	621135	XBox Interactive Video Gaming System	Games	\$159.00	XBox
☐	XBox 360 Video Game System	218407	Next generation XBox 360 Interactive Video Gaming System.	Games	\$299.00	XBox
☐	Playstation Portable	124039	Playstation Portable HandHeld Video Gaming System.	Handhelds	\$199.00	MyTag
☐	Nintendo DS	606945	Nintendo DS Handheld Gaming System	Handhelds	\$129.00	DS
☐	Razer Cellular Phone	927384	Ultra-sleek Razer Cellular Phone	Handhelds	\$229.00	Razer

When the user selects a product, and clicks the **View Details** button, the Item Details page displays the product details, as shown in Figure 2–23. This page displays detailed information about the product, and allows the users to select a quantity to add to their cart.

Figure 2–23 Item Details Page

Item Details

Name HD Television

Description A High-Definition Flat Screen Television that uses DLP Technology.

Category Video

Price \$1999.00

Product Id 128723

Producer Azimuth

Continue Shopping

Add to cart?

Quantity?

Add to cart

[Go to Shopping Cart](#)

When the user clicks the **Go to Shopping Cart** link, the Cart page displays the added items, as shown in Figure 2–24.

Figure 2–24 The SOA Order Booking Cart Page

Product Name	Part No.	Category	Price	Quantity
HD Television	128723	Video	\$1,999.00	2
Empty Shopping Cart		Continue Shopping	Place Order	Order Total: \$3,998.00

When the user clicks the **Place Order** button, the purchase order is created and sent into the OrderBookingESB flow.

Where to Find Implementation Details

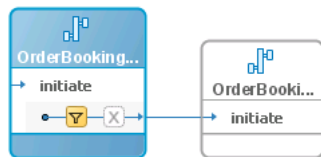
Following are the sections of the *Oracle SOA Suite Developer's Guide* that describe how to develop the web client components used in the order process:

- Displaying data using ADF data binding: The product data on the Browse Items page is displayed using ADF data binding, which allows you to declaratively bind components on a web page to your data source. For more information, see [Section 9.5, "Displaying Collection and Detail Data Using ADF Data Binding"](#).
- Invoking an ESB flow from a web page: The `ShoppingCart.java` backing bean contains a method that invokes a web service and populates the web service with the order data, in much the same way as the registration process. For more information, see [Section 9.6.3, "How to Invoke a Service Using a Backing Bean"](#).

2.5.3.2 The OrderBookingESB Flow

When the user clicks the **Place Order** button, the `OrderBookingESB` flow is invoked. That, in turn, invokes the `SOAOrderBooking` BPEL flow. [Figure 2-25](#) shows the ESB flow, as viewed in Oracle JDeveloper.

Figure 2-25 The OrderBookingESB Flow

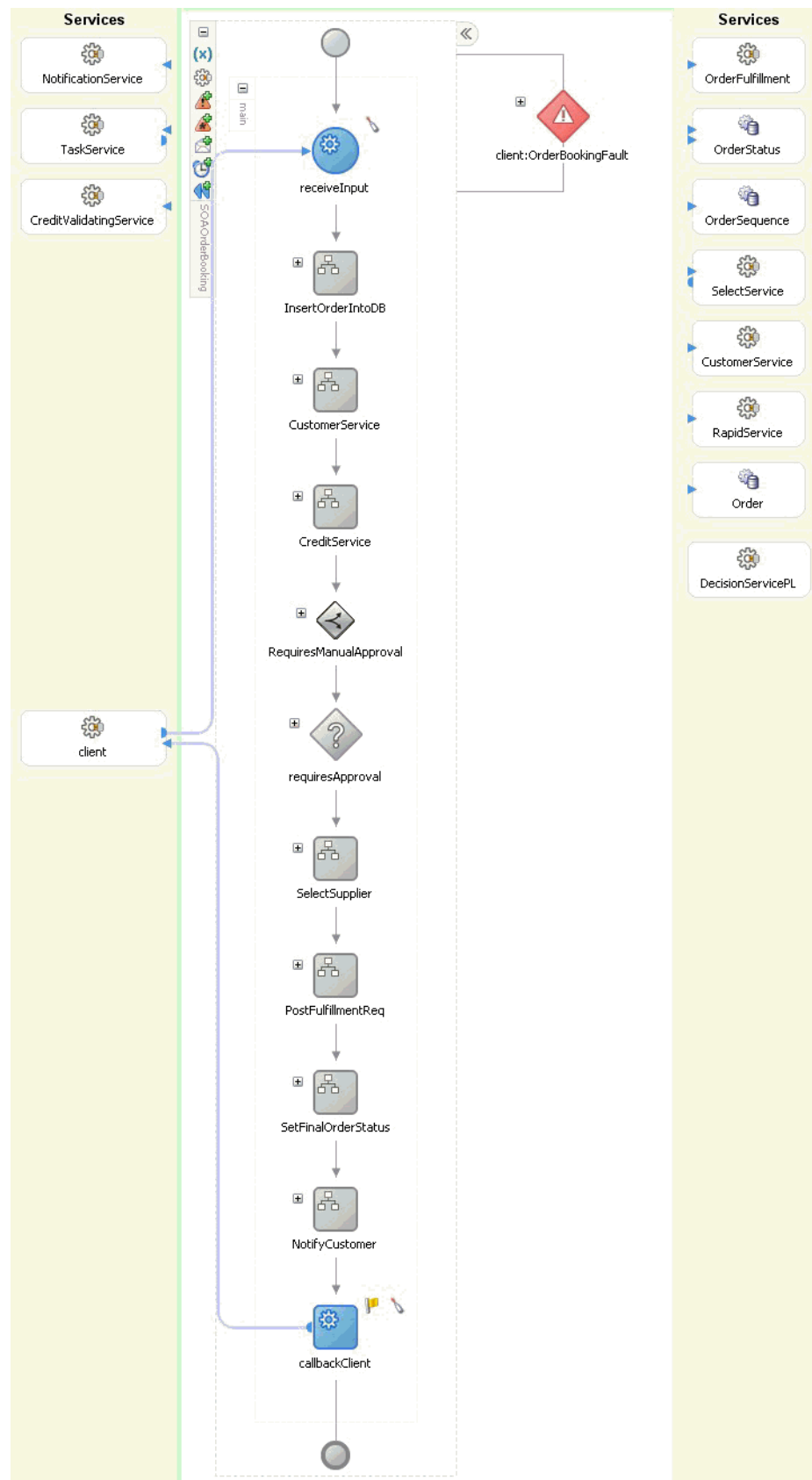


Where to Find Implementation Details

The `OrderBookingESB` flow invokes a web service for a BPEL flow. For more information, see [Section 6.7, "Adding SOAP Services to Oracle Enterprise Service Bus"](#).

2.5.3.3 SOAOrderBooking BPEL Flow

The `SOAOrderBooking` BPEL flow then handles the order process, as shown in [Figure 2-26](#).

Figure 2–26 The SOAOrderBooking BPEL Flow

Each scope activity icon (gray box with a flow) represents a higher-level view of a detailed portion of the flow. The following sections describe each step in the flow, including the details within each scope activity. The first activity (`receiveInput`) is not included, as it simply serves as the entry point into the flow.

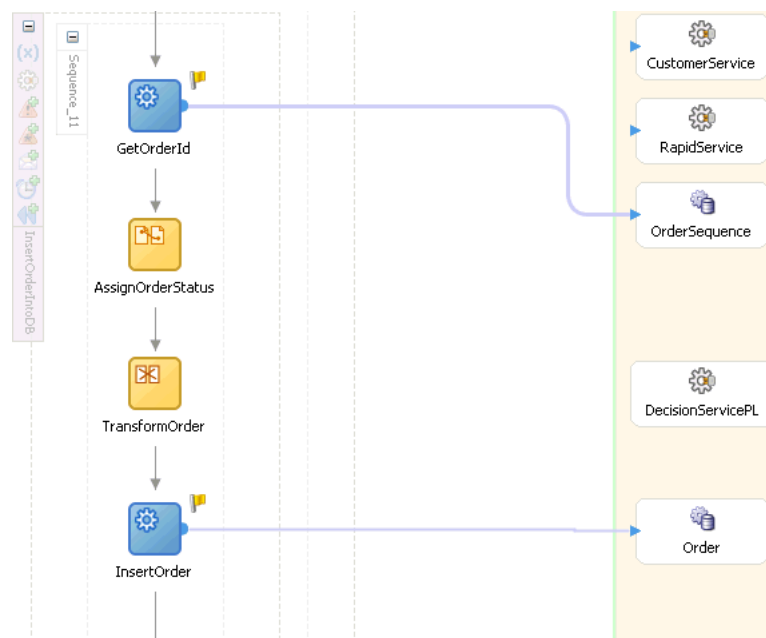
2.5.3.3.1 InsertOrderIntoDB Scope

The `InsertOrderIntoDB` scope includes the following activities:

- `GetOrderId`: Gets the order ID using a database adapter to access the `Order` sequence.
- `AssignOrderStatus`: Copies the order ID and status (`pending`) and assigns those values to variables.
- `TransformOrder`: Transforms the order data into an `orderRequest` variable.
- `InsertOrder`: Inserts the order into the `ORDER` table using a database adapter.

Figure 2-27 shows the `InsertOrderIntoDB` scope in detail.

Figure 2-27 *InsertOrderIntoDB Scope*



Where to Find Implementation Details

Following are the sections of the *Oracle SOA Suite Developer's Guide* that describe how to develop the types of components used in the `InsertOrderIntoDB` scope:

- Assigning values to variables: The `AssignOrderStatus` activity assigns the order ID number and status to variables that will be used within the scope. For more information about using and assigning variables, see [Section 7.4, "Using an Assign Activity"](#).
- Transforming data: The order data is transformed from the message content, as received from the ESB flow, to data that the database adapter accepts. For more information about transforming data, see [Section 7.5, "Creating a Transformation"](#).

- Invoking a service: The Order database service is invoked, which allows the order data to be persisted to the database. For more information about using a database adapter, see [Section 7.6, "Communicating with a Database"](#).

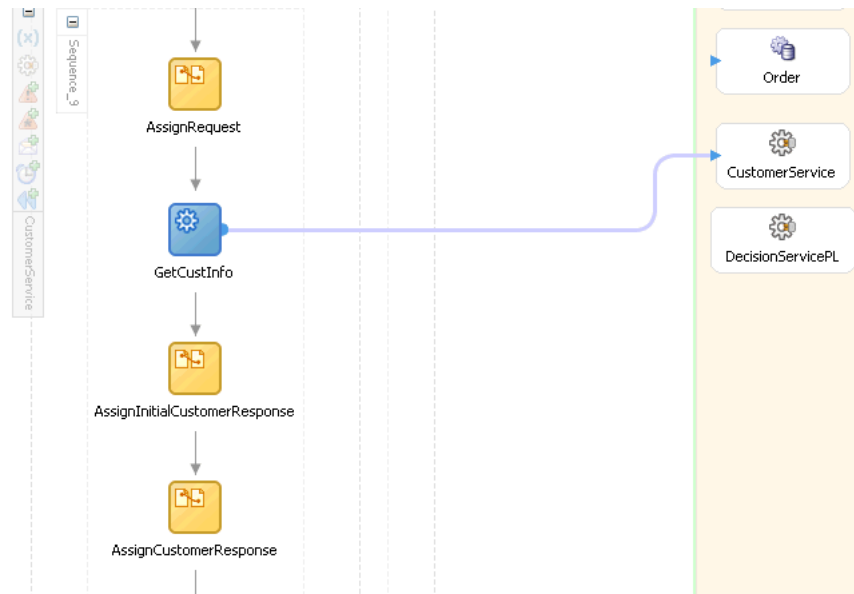
2.5.3.3.2 CustomerService Scope

The CustomerService scope includes the following activities:

- AssignRequest: Copies the customer's ID and assigns the value to a variable.
- GetCustInfo: Invokes the Customer service synchronously, using the customer ID to obtain customer information.
- AssignInitialCustomerResponse - Copies the purchase order from the ESB flow to a variable in the BPEL flow.
- AssignCustomerResponse: Copies the customer's first and last name and assigns the values to variables.

[Figure 2–28](#) shows the CustomerService scope in detail.

Figure 2–28 CustomerService Scope



Where to Find Implementation Details

Following are the sections of the *Oracle SOA Suite Developer's Guide* that describe how to develop the types of components used in the CustomerService scope:

- Synchronously invoking a service: The GetCustInfo activity uses a synchronous call to the CustomerService web service to obtain the customer's first and last name, given the customer ID. For more information about creating a synchronous call to a service, see [Section 7.3, "Invoking a Web Service"](#).
- Creating a partner link to a service: The CustomerService partner link accesses the CustomerService web service in order to retrieve customer information. For more information about creating partner links, see [Section 7.3.1.1, "Creating a Partner Link"](#).

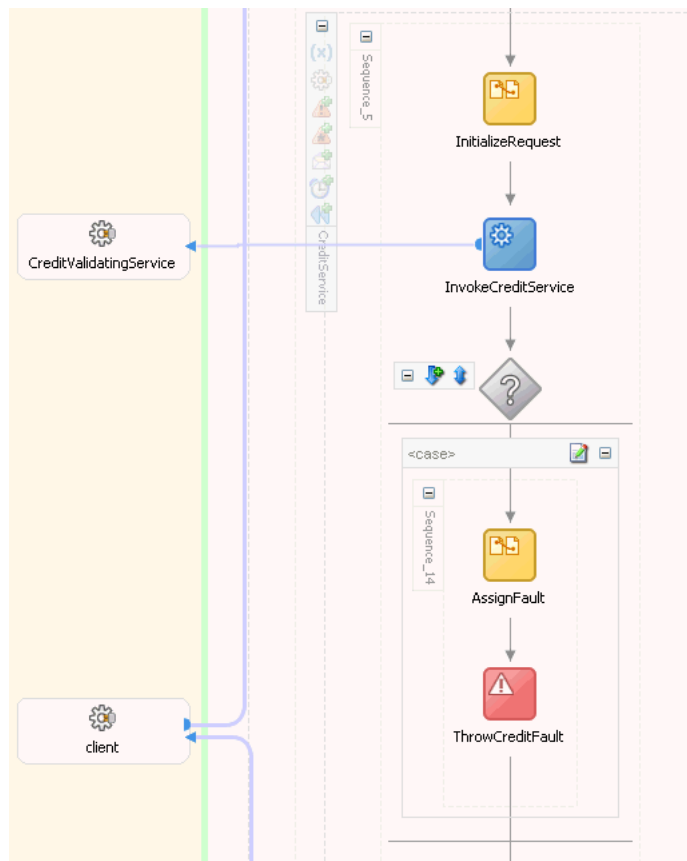
2.5.3.3.3 CreditService Scope

The *CreditService* scope includes the following activities:

- **InitializeRequest:** Copies the credit card type and number from the customer data and assigns the values to variables.
- **InvokeCreditService:** Invokes the *CreditValidatingService* web service with a synchronous call that includes the credit card number and type. The service replies with either a valid or not valid response.
- **Generic switch activity:** An expression in the switch task states that if the service returns a response stating that the card is not valid, an error must be thrown. The switch contains the following two activities:
 - **AssignFault:** Assigns the value "Credit Problem" to the status variable.
 - **ThrowCreditFault:** Invokes the *OrderBookingFault*. For more information, see [Section 2.5.3.3.11, "OrderBookingFault Scope"](#).

Figure 2–29 shows the *CreditService* scope in detail.

Figure 2–29 CreditService Scope



Where to Find Implementation Details

Following are the sections of the *Oracle SOA Suite Developer's Guide* that describe how to develop the types of components used in the `CreditService` scope:

- Creating a conditional branch using a switch activity: The switch activity allows you to create conditional cases. The flow follows that path that evaluates to `true`. For more information about creating conditional branches, see [Section 7.7, "Creating Conditional Branching Using a Switch Activity"](#).
- Handling a fault: When the credit service returns a message stating that the card is invalid, the flow must handle this as a fault, because the flow will not be able to continue. For more information about handling faults, see [Section 7.10, "Handling Faults"](#).

2.5.3.3.4 RequiresManualApproval Scope

The `RequiresManualApproval` scope includes the following activities:

- `BPEL_Header`: Gets information about the BPEL flow and assigns that information to variables.
- `BPEL_Var_To_Rule_Facts`: Assigns purchase order price and customer status variable value to facts that will be used by a rule to determine whether the order needs manual approval.
- `Facts_To_Rule_Service`: Assigns the facts to a variable.
- `Invoke`: Invokes the Decision Service, which in turn, invokes the rule to determine whether the order needs to be manually approved.
- `Rule_Service_To_Facts`: Takes the response (whether or not the order requires manual approval), and sets the value to a rule fact.
- `Facts_To_BPEL_Var`: Assigns value of the rule fact to a variable.

[Figure 2–30](#) shows the `RequiresManualApproval` scope in detail.

Figure 2-30 *RequiresManualApproval Scope***Where to Find Implementation Details**

Following are the sections of the *Oracle SOA Suite Developer's Guide* that describe how to develop the types of components used in the **RequiresManualApproval** scope:

- **Creating business rules:** The flow uses a business rule to determine whether or not an order requires manual approval. If the customer's status is platinum, no approval is required. If the customer's status is gold, and the order is over \$1,000, the order requires manual approval. If the customer is silver, the order requires manual approval. For more information about creating rules, see [Chapter 8, "Creating and Using Rules for a Decision Service"](#).
- **Using the Decision Service to invoke business rules:** While you can create rules that can be invoked throughout an application, the rules in the SOA Order Booking application are invoked from the BPEL flow using the Decision Service. For more information about using the Decision Service, see [Section 7.8, "Using Business Rules and a Decide Activity"](#).

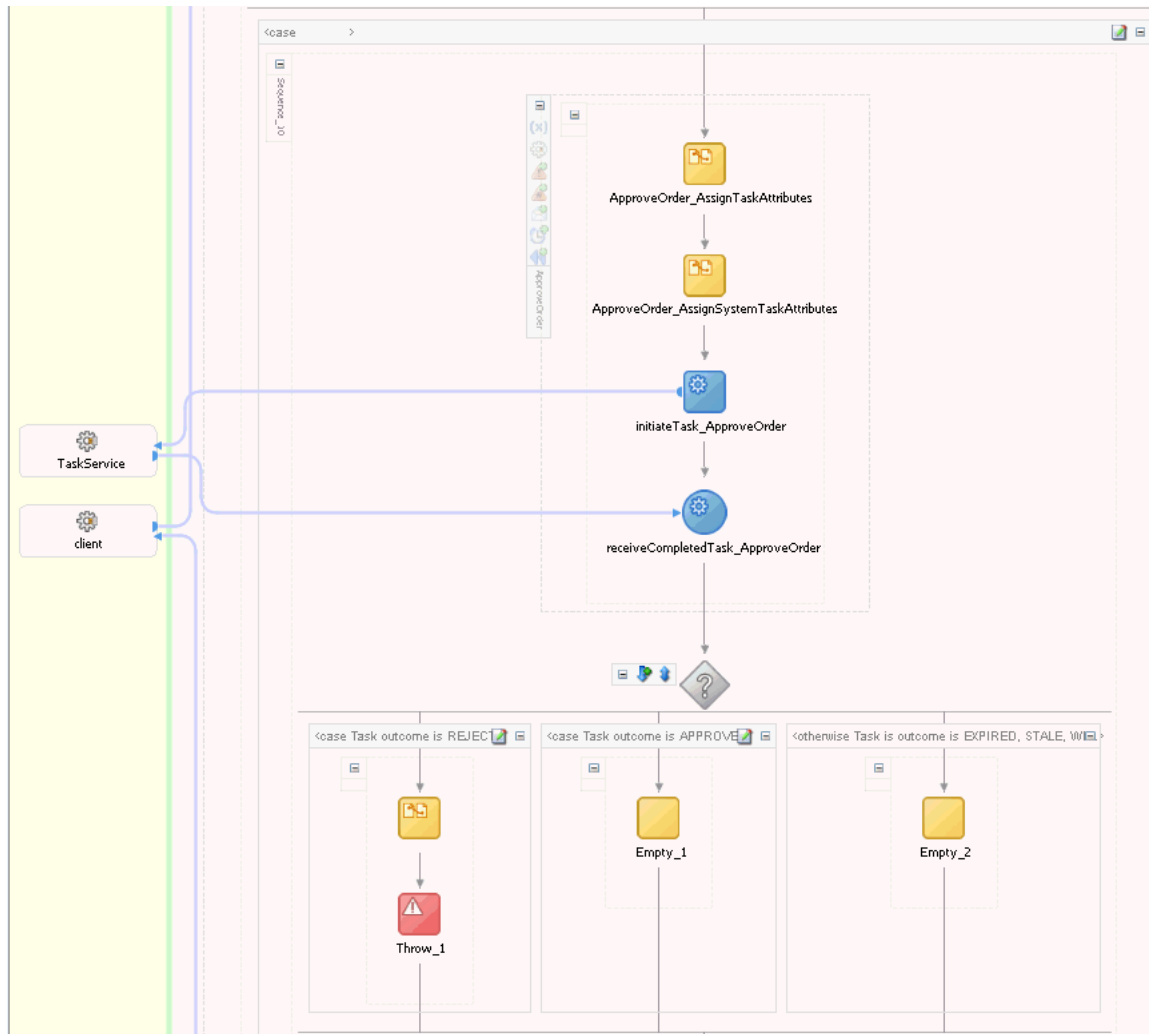
2.5.3.3.5 RequiresApproval Switch

The RequiresApproval switch is executed only if the decision service determines that the order requires manual approval. It contains the following activities:

- ApproveOrder_AssignTaskAttributes: Takes order attribute values needed to approve the order and assigns the values to variables.
- ApproveOrder_AssignSystemTaskAttributes: Takes process flow attribute values required for approval and assigns the values to variables.
- InitiateTask_ApproveOrder: Invokes a human workflow with an asynchronous call to the TaskService web service.
- ReceiveCompletedTask_ApproveOrder: The response from the human workflow is received.
- Switch task: Determines the flow based on whether or not the order was approved.
 - Outcome is Reject: If the order was rejected, the flow throws the Order Booking fault.
 - Outcome is Approve: If the order was approved, the flow simply continues.
 - Otherwise: Used if the task outcome is expired, stale, withdrawn, or rejected.

[Figure 2–31](#) shows the RequiresApproval switch in detail.

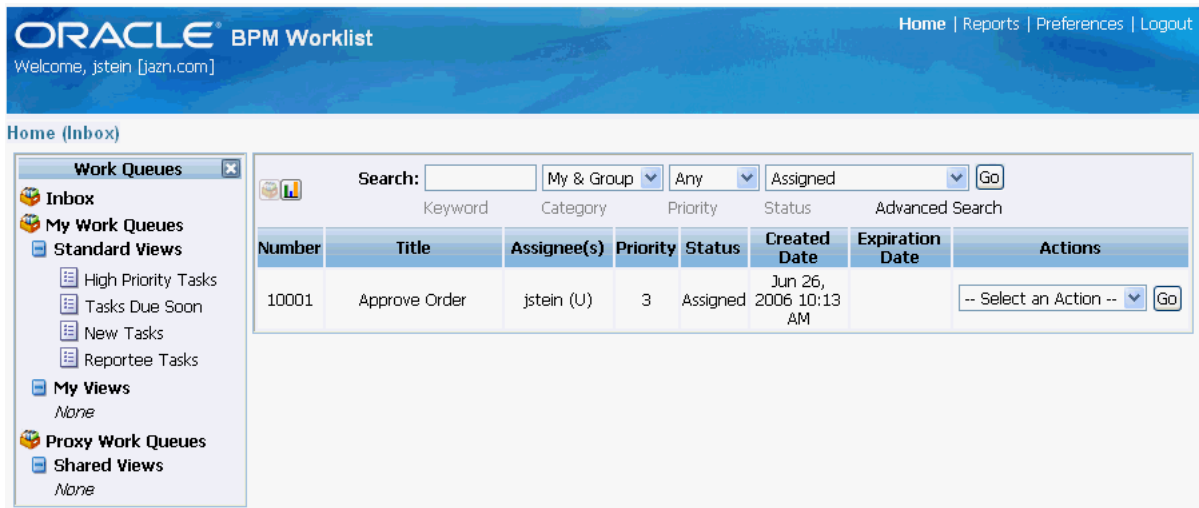
Figure 2–31 *RequiresApproval Switch*



Where to Find Implementation Details

When an order requires manual approval, the BPEL flow invokes the `TaskService` web service to start the human workflow to approve the order. This service is invoked asynchronously, sending information into the workflow. This workflow provides a web-based interface that allows a user to complete the assigned task. [Figure 2–32](#) shows the web page that allows the manager to approve the order. Once the order is approved, information is sent back into the BPEL flow, again using an asynchronous call. For more information about using the Task Service, see [Section 7.12, "Creating a Human Workflow Task"](#).

Figure 2–32 Workflow Web Page



2.5.3.3.6 SelectSupplier Scope

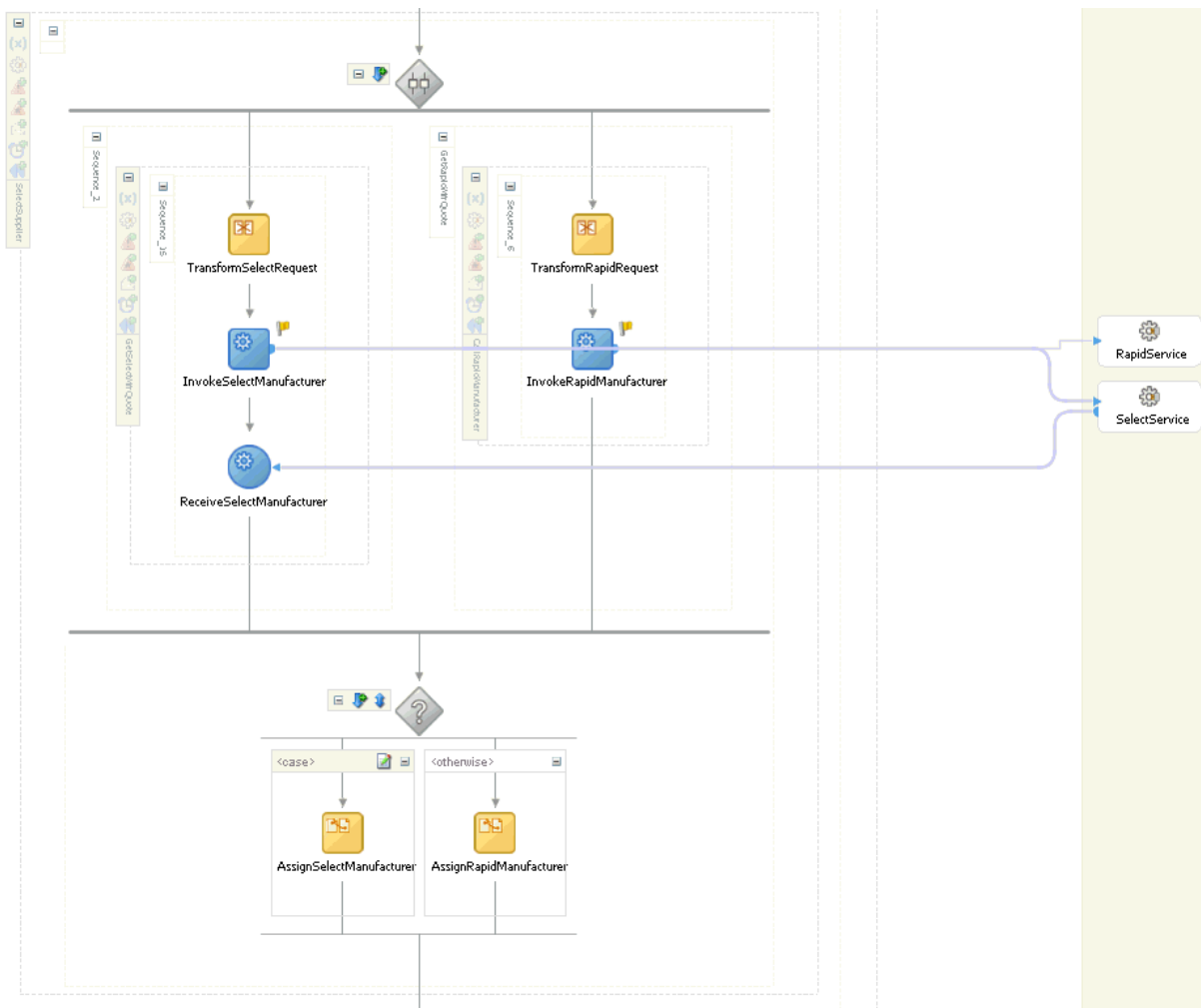
The `SelectSupplier` scope contains parallel flows that send information to two different suppliers. The suppliers each return a price for the order. A switch activity then determines which supplier will be used to fulfill the order based on an evaluated condition. The `SelectSupplier` scope includes the following activities:

- `GetSelectMfrQuote`: A flow that sends order information in an asynchronous call to `Select Manufacturer`. This flow includes the following activities:
 - `TransformSelectRequest`: Takes order information and transforms it into parameters needed by `Select Manufacturer` in order to complete a quote.
 - `InvokeSelectManufacturer`: Invokes the `SelectService` web service, sending the information as transformed in the previous activity.
 - `ReceiveSelectManufacturer`: Receives the asynchronous response from `Select Manufacturer` and stores it as a variable.
- `CallRapidManufacturer`: A flow that sends order information in a synchronous call to `Rapid Manufacturer`. This flow includes the following activities:
 - `TransformRapidRequest`: Takes order information and transforms it into parameters needed by `Rapid Manufacturer` in order to complete a quote.
 - `InvokeRapidManufacturer`: Invokes the `RapidService` web service to obtain a quote from `Rapid Manufacturer`.

- **SelectByPrice Switch:** An expression determines which manufacturer will be assigned the order. If the Select Manufacturer price is less than the Rapid Manufacturer price, the order is assigned to Select Manufacturer. Otherwise, it is assigned to Rapid Manufacturer. Depending on the outcome, one of the following activities is invoked:
 - **AssignSelectManufacturer:** Assigns the value for the price and Select Manufacturer to variables.
 - **AssignRapidManufacturer:** Assigns the value for the price and Select Manufacturer to variables.

Figure 2–33 shows the SelectSupplier scope in detail.

Figure 2–33 SelectSupplier Scope



Where to Find Implementation Details

Following are the sections of the *Oracle SOA Suite Developer's Guide* that describe how to develop the types of components used in the `SelectSupplier` scope:

- Creating parallel flows: The two flows that call out to the suppliers happen simultaneously. For more information about creating parallel flows, see [Section 7.9, "Creating a Parallel Flow"](#).
- Asynchronously invoking a service: The flow invokes the `SelectService` web service using an asynchronous call. The service then sends a separate reply back into the flow.

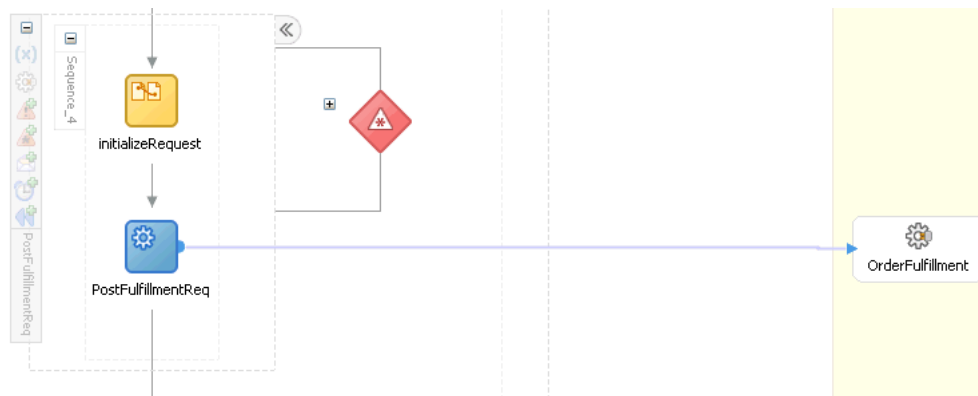
2.5.3.3.7 PostFulfillmentReq Scope

The `PostFulfillmentReq` scope invokes an ESB flow used to determine the shipping vendor. It contains the following activities:

- `InitializeRequest`: Assigns the value of a variable that represents the purchase order to a variable that will be used in an ESB flow.
- `PostFulfillmentReq`: Invokes the `OrderFulfillment` ESB flow, passing in the variable value. For more information about this ESB flow, see [Section 2.5.3.4, "OrderFulfillmentESB Flow"](#).

[Figure 2–34](#) shows the `PostFulfillmentReq` scope in detail.

Figure 2–34 The PostFulfillmentReq Scope



Where to Find Implementation Details

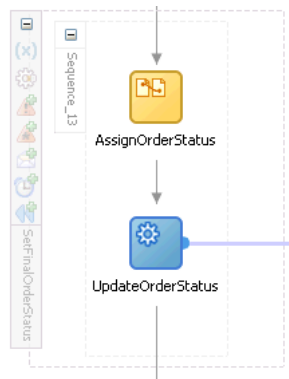
The `PostFulfillmentReq` activity invokes an ESB flow. For more information about invoking an ESB flow, see [Section 7.3, "Invoking a Web Service"](#).

2.5.3.3.8 SetFinalOrderStatus Scope

The `SetFinalOrderStatus` scope uses a database adapter to update the status of the order in the `ORDERS` table. The `SetFinalOrderStatus` scope includes the following activities:

- `AssignOrderStatus`: Copies the order ID and status (completed) and assigns those values to variables.
- `UpdateOrderStatus`: Invokes the database adapter and uses the variables to update the status of the order.

[Figure 2–35](#) shows the `SetFinalOrderStatus` in detail.

Figure 2–35 SetFinalOrderStatus Flow**Where to Find Implementation Details**

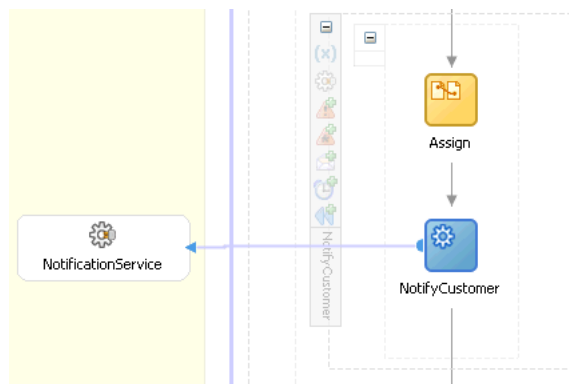
The `Order` database service is invoked, which allows the order data to be persisted to the database. For more information database adapters, see [Section 7.6, "Communicating with a Database"](#).

2.5.3.3.9 NotifyCustomer Scope

The `NotifyCustomer` scope invokes the Notification service to let customers know that their order has been fulfilled. The `NotifyCustomer` scope includes the following activities:

- **Assign:** Assigns order and customer variable values to variables that will be used by the Notification service to create the email.
- **NotifyCustomer:** Invokes the Notification service to send the email.

[Figure 2–36](#) shows the `NotifyCustomer` scope in detail.

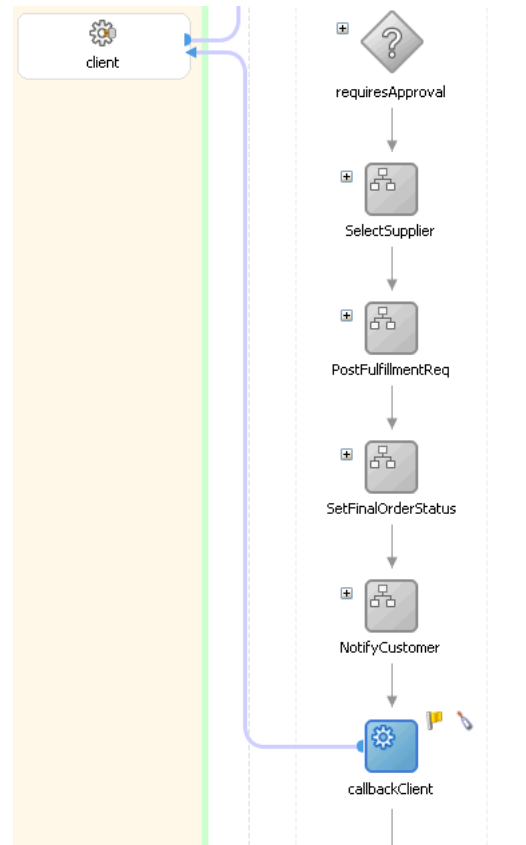
Figure 2–36 NotifyCustomer Scope**Where to Find Implementation Details**

The `NotifyCustomer` activity invokes the Notification service, passing in variables that will be used to create an email notification. The Notification service then sends the email. For more information about the Notification service, see [Section 7.13, "Creating an Email Notification"](#).

2.5.3.3.10 CallbackClient Activity

The `CallbackClient` activity invokes the web client, as shown in [Figure 2–37](#). This activity sends a response message back to the client, with the status of the order.

Figure 2–37 *CallbackClient Scope*

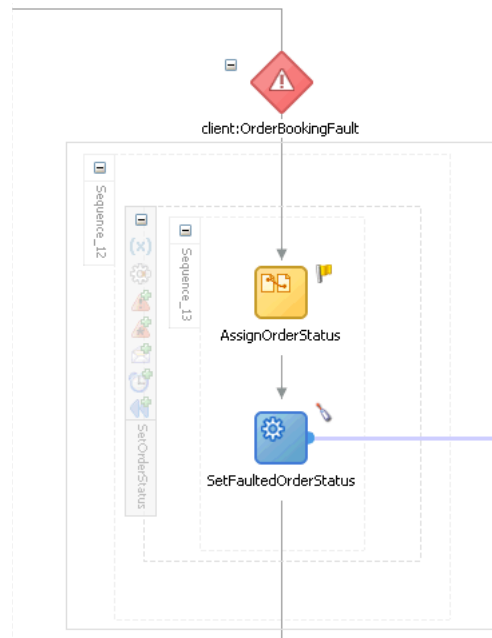


2.5.3.3.11 OrderBookingFault Scope

The `OrderBookingFault` scope is a generic fault that is invoked whenever a fault is encountered in the flow. The `OrderBookingFault` scope includes the following activities:

- `AssignOrderStatus`: Assigns variable values for the order and status.
- `SetFaultedOrderStatus`: Sets the status of the order to `Cancelled`, using a database adapter.

[Figure 2–38](#) shows the `OrderBookingFault` scope in detail.

Figure 2–38 OrderBookingFault**Where to Find Implementation Details**

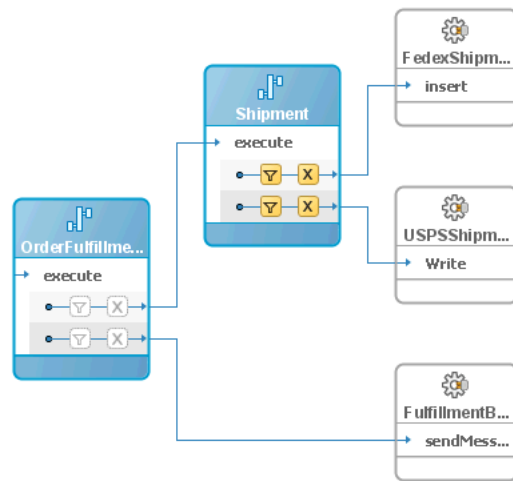
The `OrderBookingFault` scope is used to handle all exceptions and errors thrown in the process flow. For more information about creating a fault for the flow, see [Section 7.10, "Handling Faults"](#).

2.5.3.4 OrderFulfillmentESB Flow

The `OrderFulfillmentESB` flow determines which vendor (USPS or Federal Express) will ship the order. The determination is made based on routing rules. This flow includes the following services:

- `OrderFulfillment` routing service: Routes order information to the `Shipment` routing service. It also routes a message to the `FulfillmentBatch` JMS adapter.
- `Shipment` routing service: Routes order information to FedEx when the order total is greater than \$500. Routes order information to the USPS when the order is less than \$500. It then transforms the information into a message understood by FedEx or USPS, as appropriate.
- `FedexShipment` database adapter: Sends order information to FedEx, and inserts it into the `FEDEXSHIPMENT` database table.
- `USPSShipment` database adapter: Sends order information to USPS.
- `FulfillmentBatch` JMS adapter: Uses the JMS adapter to send a completion message back into the BPEL flow.

[Figure 2–39](#) shows the `OrderFulfillmentESB` flow in detail.

Figure 2–39 OrderFulfillmentESB Flow**Where to Find Implementation Details**

Following are the sections of the *Oracle SOA Suite Developer's Guide* that describe how to develop the components used in the OrderFulfillmentESB flow:

- Routing messages to endpoints: The Shipment routing service is used to determine which vendor to use based on a set of rules. For information about adding a routing service to ESB, see [Section 6.9, "Adding Routing Services to Oracle Enterprise Service Bus"](#).
- Populating database tables: Database adapter services can be used to populate or query database tables. For more information, see [Section 6.8, "Adding Adapter Services to Oracle Enterprise Service Bus"](#).
- Placing messages on a JMS queue or topic: JMS adapter services can be used to consume or produce messages and place them in the JMS queue. For more information, see [Section 6.8, "Adding Adapter Services to Oracle Enterprise Service Bus"](#).

2.5.4 Adding Security to the System

While the SOA Order Booking application does not incorporate security, you can use Oracle WSM to protect web services and process flows in a service-oriented environment, without having to modify either client applications or web services.

Oracle WSM secures your services environment with these key components:

- Oracle WSM Policy Manager: Allows you to define policies that reflect operational best practices and requirements. It includes a browser-based tool for creating and maintaining security and management policies for web services and business processes, using prebuilt or custom policy steps
- Oracle WSM Gateways: Gateways provide a nonintrusive mechanism for policy enforcement.
- Oracle WSM Agents: Also serving as policy enforcement points, agents are lightweight components that run in the same "container" or application server environment as the web service. Oracle Web Services Manager provides two types of agents, a client agent and a server agent:

- Oracle WSM Monitor: The Oracle WSM Monitor component manages collection and aggregation of web services traffic data and provides alerts and notifications. For an overview of the monitor, see [Section 2.5.5.4, "Oracle Web Services Manager Monitor"](#).

For more information about using Oracle WSM, see [Chapter 10, "Securing the System"](#).

2.5.5 Monitoring the System

Once an order is in-flight, you can monitor its process through the ESB and BPEL process flow using Oracle Enterprise Service Bus Control and Oracle BPEL Control. You can also view the status of the system in general, using Oracle Enterprise Manager. If you apply security to any services in the SOA system, you can monitor that security using Oracle Web Services Manager Control Console.

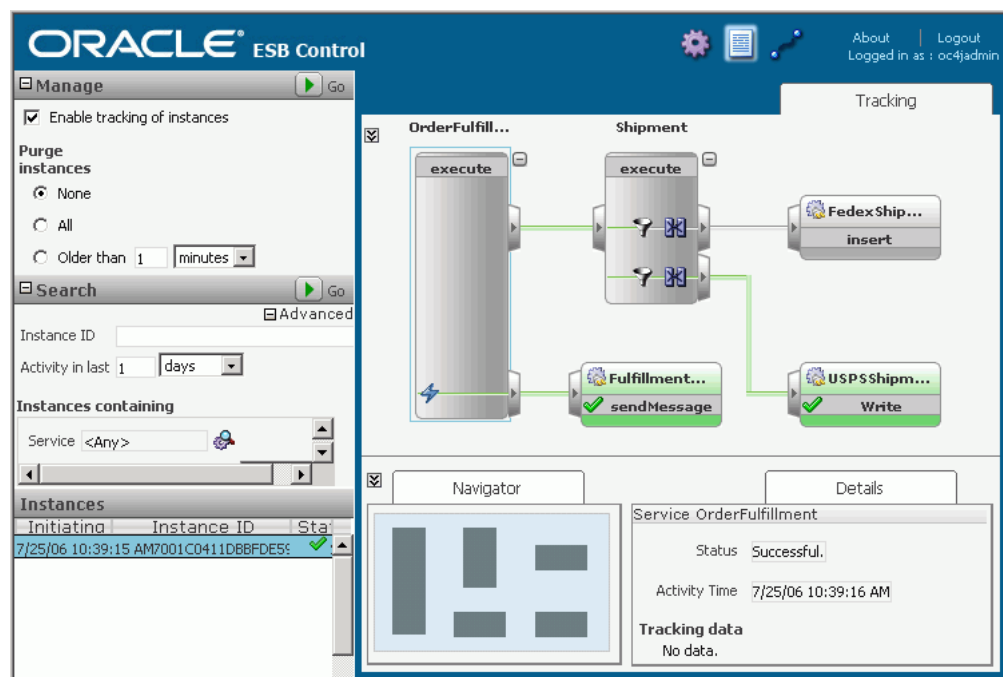
2.5.5.1 Oracle ESB Control

Oracle ESB Control is a browser-based tool that allows you to monitor message instance processing across an ESB flow in a schematic diagram that is a representation of the ESB flow. Using this diagram, you can:

- View all connections to the ESB
- View details of the ESB, including the URLs required to call ESB services from applications that are external to ESB
- View error conditions within a schematic, including any messages that did not reach their destination
- Correct error conditions, including resubmitting failed messages

[Figure 2–40](#) shows the ESB Control as it might be used to view a successful message processing.

Figure 2–40 ESB Console - Instances View



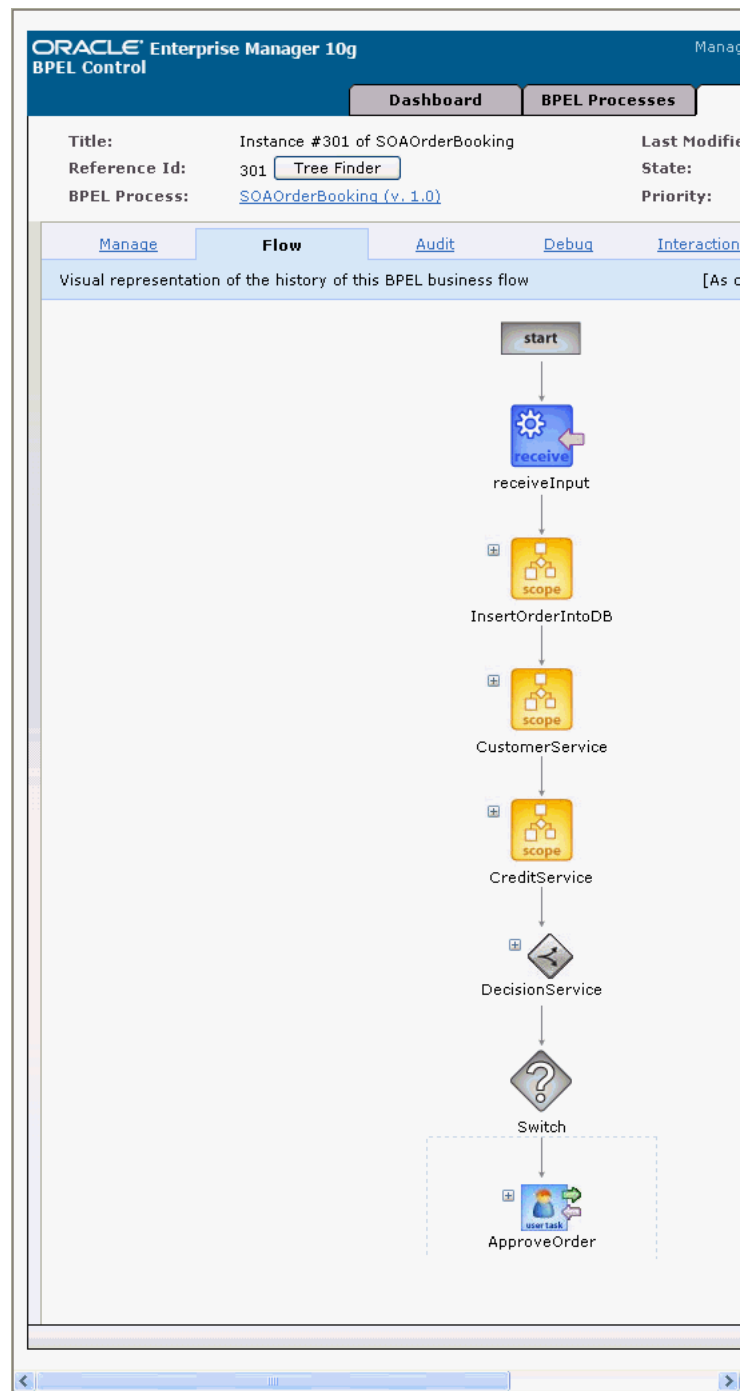
For more information about using Oracle ESB Control, see [Chapter 12, "Monitoring Oracle Enterprise Service Bus"](#).

2.5.5.2 Oracle BPEL Control

Oracle BPEL Control is a browser-based tool for managing the life cycle of a BPEL process. Using Oracle BPEL Control, you can:

- View deployed, running, and completed processes
- Manage the process life cycle, including recovery of any failed messages within an instance, and initiate processes
- Enter test data and directly initiate processes, thus allowing you to test processes before deploying to the SOA system
- View process instances
- View process activities

[Figure 2-41](#) shows the BPEL Control displaying an `SOAOrderBooking` process instance, as it waits for manual approval.

Figure 2–41 SOA Order Booking Instance in Oracle BPEL Control

For more information about using Oracle BPEL Control, see [Chapter 13, "Monitoring Oracle BPEL Process Manager"](#).

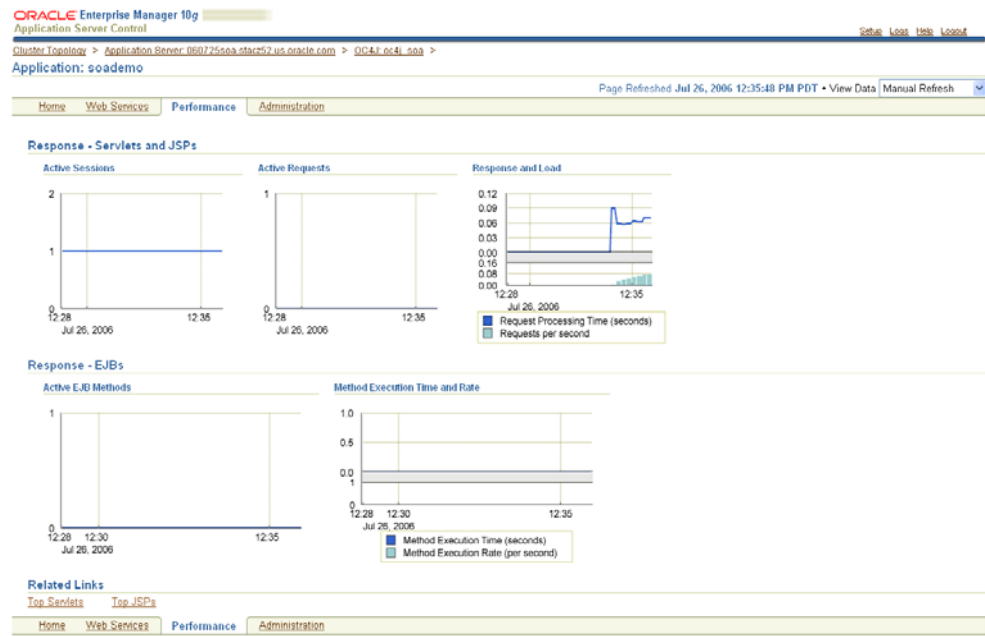
2.5.5.3 Oracle Enterprise Manager Control Console

Oracle Enterprise Manager 10g Application Server Control Console is a web-based user interface that allows you to monitor applications within OC4J. Using Enterprise Manager, you can monitor:

- An OC4J instance
- The cluster topology
- Deployed applications
- Web module and EJB module performance
- JVM metrics

Figure 2–42 shows Enterprise Manager as it might be used to monitor performance of the J2EE container.

Figure 2–42 Container Performance



For more information about using Enterprise Manager to monitor the system, see [Chapter 15, "Monitoring SOA Applications with Enterprise Manager"](#).

2.5.5.4 Oracle Web Services Manager Monitor

Oracle Web Services Manager Monitor is a browser-based tool that allows you to monitor the status and performance of the managed services used within your SOA system, including extensive features that enable you to monitor the operational health of the components (gateways and agents) that you set up to secure your web services.

Using Web Services Manager Monitor, you can:

- View execution metrics
- View security statistics
- View service statistics
- View alarms
- Manage security roles

Figure 2–43 shows Web Services Manager Monitor, as it might be used to monitor security statistics.

Figure 2–43 Monitoring Security

The screenshot displays the Oracle Enterprise Manager 10g Web Services Manager Control interface. The left sidebar contains navigation links: Policy Management, Operational Management (selected), Snapshot, Overall Statistics, Security Statistics (selected), Access Control, Service Statistics, My Views, Alarms, and Alarm Rules. The main content area shows the 'Access Control' page with a breadcrumb trail: Operation Management > Security Statistics > Access Control. Below the breadcrumb, there is a 'Help' link and a section titled 'Access Control Display Criteria'. This section includes three dropdown menus: 'Component' set to 'Authentication Agent', 'Service' set to 'ALL SERVICES', and 'Time Range' set to 'Last hour'. A 'Display' button is located below these dropdowns. At the bottom, there is a table titled 'Access Control' with the following data:

Component	Service	Start	End
Authentication Agent	ALL SERVICES	Friday, August 4, 2006 11:48:00 AM	Friday, August 4, 2006 12:48:00 PM

Part II

Developing SOA Systems

Part II contains the following chapters:

- [Chapter 3, "Designing an SOA Application"](#)
- [Chapter 4, "Building and Using Application Services"](#)
- [Chapter 5, "Building Web Services"](#)
- [Chapter 6, "Using Oracle Enterprise Service Bus"](#)
- [Chapter 7, "Using Oracle BPEL Process Manager"](#)
- [Chapter 8, "Creating and Using Rules for a Decision Service"](#)
- [Chapter 9, "Developing a Web Application in an SOA System"](#)
- [Chapter 10, "Securing the System"](#)
- [Chapter 11, "Deploying SOA Applications with Enterprise Manager"](#)

Designing an SOA Application

This chapter describes how to successfully design an SOA application using a methodology tailored for SOA projects. The design process discussed in this chapter is based on the Oracle SOA Success Methodology, used by Oracle consultants for rapid delivery of enterprise SOA systems. The result of following this methodology will be the definition and design of an SOA solution that can be used in solution implementation. This chapter provides high-level information only, and is not meant to provide a complete methodology for designing an SOA project. Additionally, it focuses on the supplemental steps required in designing an SOA solution, and does not cover standard application planning and design. While it is based on the Oracle SOA Success Methodology, the principals are key to any successful SOA design strategy and can be easily adapted to any methodology you may currently have in place.

This chapter includes the following sections:

- [Section 3.1, "Introduction to Designing an SOA Application"](#)
- [Section 3.2, "Defining the Project in the Project Scope"](#)
- [Section 3.3, "Designing the Project in the Application Scope"](#)

3.1 Introduction to Designing an SOA Application

To achieve the main benefits of an SOA system, such as reusability and agility, you must pay careful attention during design to ensure that those benefits are realized. SOA begins at the design phase, where you must deliberately plan with the SOA paradigm in mind. Key design tactics to achieve this include:

- Creating a service portfolio plan, where you keep track of all services used within an enterprise (to ensure reuse where possible)
- Ensuring that objectives and requirements reflect SOA benefits, such as reuse and loose-coupling
- Limiting dependencies by using business processes to orchestrate services instead of hardcoding dependencies within components
- Using design patterns to regulate best practices and standards

The Oracle SOA Success Methodology defines the following three scopes within which SOA solution design takes place:

- **Enterprise scope:** This scope encompasses the larger SOA framework within which applications for an enterprise exist. For example, Global Company (the fictional company associated with the SOA Order Booking application) may have systems other than order booking, such as accounting applications, sales tracking applications, and so on. All these applications, including the order booking application, reside within the enterprise scope.

It is within the enterprise scope that a business determines that adapting an SOA paradigm makes sense. At this level, such things as return on investment (ROI) models and SOA road maps are created and updated. It is also in this scope that a business should develop an inventory or portfolio of all existing applications and functionality. Doing so encourages reuse or modification of existing functionality, a fundamental aspect of the SOA paradigm. For more information about creating a portfolio plan, see [Section 3.2.1, "Creating a Service Portfolio Plan"](#).

- **Project scope:** This scope is nested within the enterprise scope, and encompasses the scope for a group of applications that work together to provide a solution for a business need. The applications that make up the SOA Order Booking application fall into the Order Booking's project scope.

Some applications may appear in more than one project scope. For example, the SOA Order Booking application interacts with a customer service application that stores basic customer information. In an actual retail business, this might be a larger application that is also accessed by the sales tracking application. The sales tracking and order booking applications do not themselves interact, and so are not in the same project scope. However, the customer service application would appear in each of their respective project scopes. This type of reuse is the cornerstone of the SOA paradigm.

It is within the project scope that solution definition takes place. Here, concrete goals are set for the project, business and technical requirements are gathered, and inventories are taken to determine what needed functionality currently exists within existing applications, what existing applications may need modifications, and what applications will need to be developed. For more information about the work done within the project scope, see [Section 3.2, "Defining the Project in the Project Scope"](#).

- **Application scope:** This scope is nested within the project scope, and encompasses a single application or service. Included in this scope are logical and then physical designs for the disparate applications, services, processes, and so on. that make up an SOA project. For more information about the work done within the project scope, see [Section 3.3, "Designing the Project in the Application Scope"](#).

The outermost scopes deal mostly with planning, while the innermost scopes deal mostly with execution. For example, the enterprise scope deals only with planning, while the application scope is where implementation takes place.

This chapter covers the methodology for use within the project and application scopes, as it is assumed the acceptance of SOA in the larger enterprise scope has already been accomplished. Read this chapter to understand:

- How to create a service portfolio plan
- The work involved in the project scope
- The work involved in the application scope

3.2 Defining the Project in the Project Scope

You start the definition and design process within the project scope. In this scope, you assess what the system should achieve, the system's requirements (both business and technical), and the applications and services that will be needed to satisfy those requirements. At the end of this phase, you develop a solution specification that details the findings in this phase, and provides the necessary information to design and implement the individual applications in the application scope.

When designing the solution definition for the project, you take the following steps:

1. Create a service portfolio plan.

While technically part of the enterprise scope, the service portfolio plan is a collection of all services an enterprise currently has or will need. Creating and then maintaining this portfolio allows an enterprise to reuse existing services, thus reducing software and data redundancy. For more information, see [Section 3.2.1, "Creating a Service Portfolio Plan"](#).

2. Set business and system objectives.

When setting business objectives, you determine what the end result should be, without any regard to what may already exist. Your focus is on the solution of a business problem. Once set, these goals are used to drive requirements. For more information, see [Section 3.2.2, "Determining Business and System Objectives"](#).

3. Determine key business and operational requirements

While best practices for gathering requirements for an SOA project are the same as for any type of application or application integration project, there are some additional steps necessary to ensure that SOA benefits are fully realized. For example, the requirements for each service or piece of functionality in the project must take into account all parts of the project, and even services outside the project scope, and not just the service itself.

To achieve this, you design use cases for SOBAs (service-oriented business applications), which are large-grained use cases that encompass a specific business process. For example, a registration process might be considered a SOBA. Once you define a SOBA use case, you then focus on the use cases for each step within the SOBA's process, and the services each step uses. For more information, see [Section 3.2.3, "Determining Business and Operational Requirements"](#).

4. Conduct a discovery phase

This phase first includes analysis of the requirements in the previous step to determine a high-level design of the solution, without identifying any actual products or services that might be part of the solution. It then includes actual service discovery, where it is determined what existing services might provide needed functionality, what existing services might need to be modified, and what services will need to be created. For more information, see [Section 3.2.4, "Conducting a Discovery Phase"](#).

5. Define the business process orchestration layer and the services it invokes.

Once you have determined the business flows and services that need to be implemented, you begin defining how they will integrate. In the previous steps, the focus was on the high-level business needs. In this step, the focus shifts to technical needs. For example, in this step you define the business process model logic, process variable definitions, and more detailed exception handling. For services, you define the responsibilities of the services, along with their needed granularity. This definition provides a more precise understanding of the requirements and provides details on the internal workings of the system. For more information, see [Section 3.2.5, "Defining the Project"](#).

6. Create a solution specification that can then be used in the application scope of the project. For more information, see [Section 3.2.6, "Creating a Solution Specification"](#).

3.2.1 Creating a Service Portfolio Plan

You create a service portfolio plan in order to rationalize acquiring or developing new software applications. By creating and maintaining this plan, you ensure that existing services and applications are reused or repurposed whenever possible. This plan also reduces the time spent in creating needed functionality.

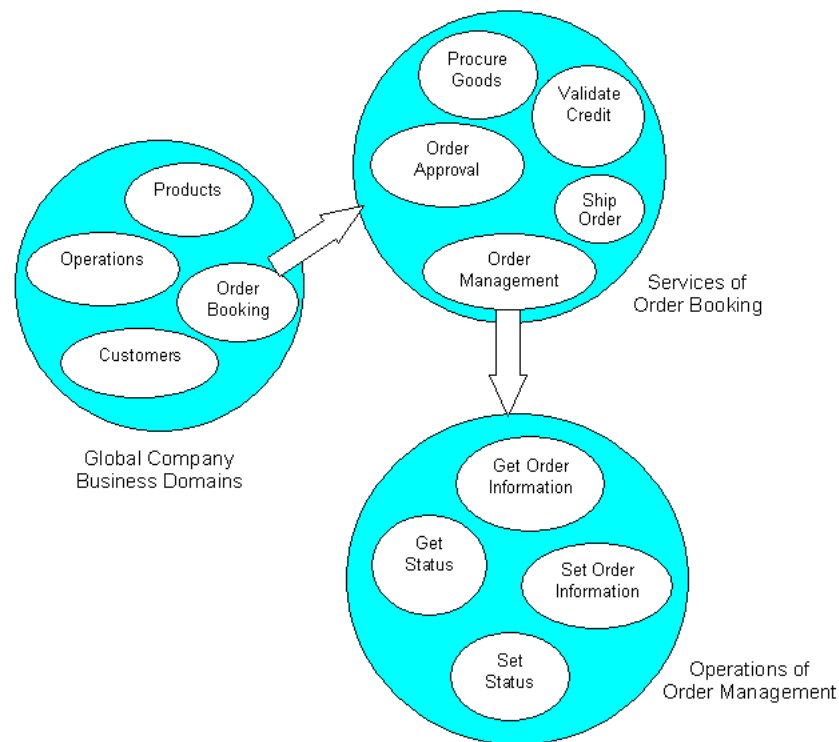
To create a service portfolio plan:

1. Conduct a top-down analysis of needed functionality (both existing and planned) for the enterprise.

To do this, execute your domain decomposition by restating the enterprise in terms of high-level business domains. Then you define services and their operations within each domain.

For example, Global Company sells electronics to consumers. In the past, they've conducted business through mail-order. Customers called into customer service representatives, who then entered customer and order information into the system. Much of the order process, such as credit validation and determining whether an order needed management approval, required swivel-chair (or manual) tasks. Global Company now wants their customers to be able to purchase products directly, using a web-based application. This means that many of the swivel-chair tasks must now be automated. In addition, Global Company would like to share the automated order booking functionality between the web system and the system used by the internal representatives.

To achieve this, Global Company performs a domain decomposition, as shown in [Figure 3-1](#). This figure shows some of the high-level business domains, including Products, Operations, Order Booking, and Customers. For the Order Booking domain, it is determined that there are a number of services required to place an order, including procuring goods, validating credit, approving orders, order management, and shipping orders. It is further determined that the order management service might contain operations for getting and setting order information, and getting and setting status.

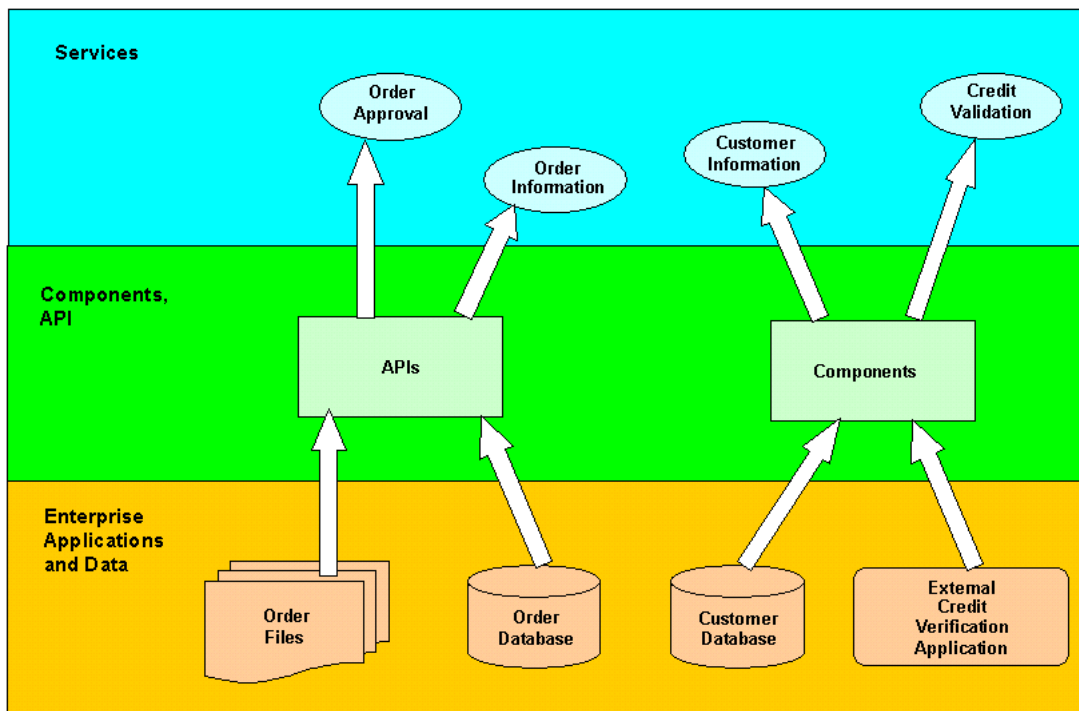
Figure 3–1 Partial Domain Decomposition for Global Company

The listing of services for each domain becomes the first version of the service portfolio.

2. Conduct a bottom-up analysis of existing applications and services.

To do this, you review all existing data and functionality currently in use, then aggregate them into groups that provide functionality for a specific task, and assign service descriptions to the aggregates. Typically, when this step is performed, you'll find additional services that were not captured in the top-down analysis.

For example, [Figure 3–2](#) shows a partial listing of the existing aggregated services before Global Company created the SOA Order Booking application:

Figure 3–2 Partial Bottom-Up Analysis of Global Company

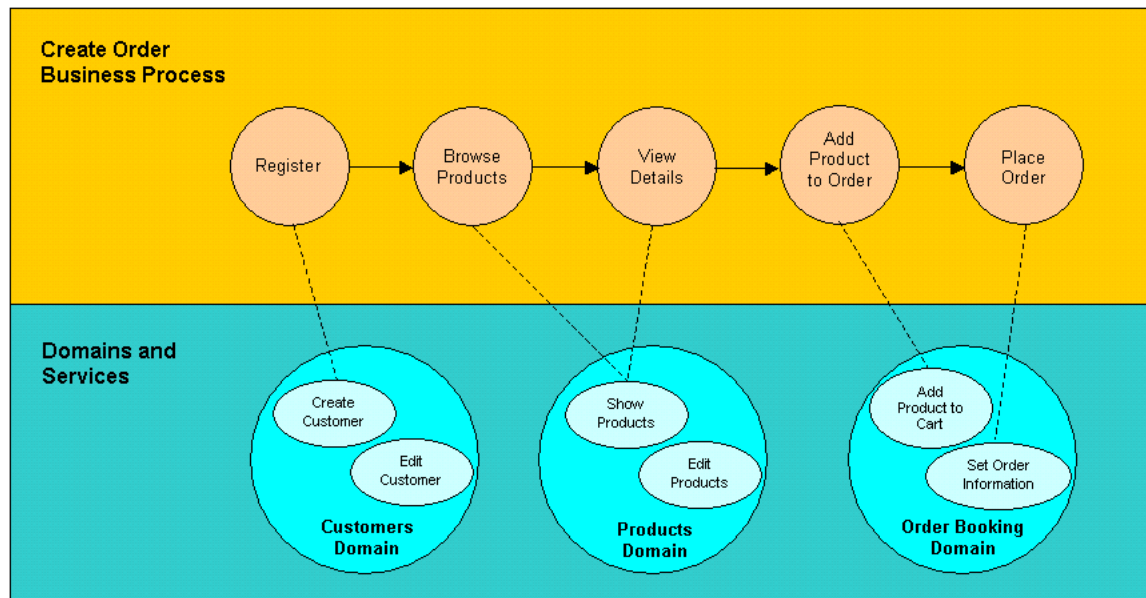
Note that the Credit Validation service was discovered in this analysis. Very often, processes that involve systems external to the company are overlooked during the top-down analysis, and then discovered in the bottom-up analysis.

Tip: Be sure to include in the portfolio any services necessary to communicate with applications external to the company. For example, the Order Booking project requires communication with a credit validation that is external to Global Company. This service should be included in Global Company's portfolio.

3. Conduct a business process tracing.

In this step, you note the business processes currently in place or needed within the enterprise. To do this, you trace a high-level business process to identify interactions within the environment that the process executes. You then collect the key events that this business process generates, and the responses that it expects to receive from the environment because of those events. These responses form the basis of operations that some service must provide.

For example, [Figure 3–3](#) shows the business process for shopping in the web-based portal, and the associated operations that some services must provide. In order to create an order, the user must register. The user can then browse through the products, view details of the products, add any product to an order, and then place the order. This order process requires specific services to provide the needed functionality. Note that services contained in one domain are often not used in the same processes.

Figure 3–3 Business Process Tracking for Creating an Order

During this tracking process, it's discovered that cart functionality is needed. That functionality is added to the portfolio as a conceptual service.

4. Iterate over steps 1, 2, and 3, refining the list of services. Note which services currently exist, and which will be needed to complete the portfolio.

The portfolio plan should be revisited each time any project is completed, to ensure that all functionality created during a project is present in the portfolio.

3.2.2 Determining Business and System Objectives

Project objectives, when realized, solve a business problem. Well-defined objectives are measurable and often relate directly to business processes and deliverables. For example, a partial list of Global Company's business objectives might be to:

- Allow 80% of customers to create their own orders using a web site
- Reduce by 50%, the amount of time customer service employees spend entering orders
- Raise customer service response time by 75% by allowing them to view real-time order status

In an SOA project, the focus should be on business benefits. Because of this, system objectives (those that are not based on functionality) should be defined only where they deliver business benefits by supporting one or more business goals. For example, Global Company might have the following system goals that support the goal of a self-service web site:

- The system should be available to users 24 hours a day, 7 days a week.
- System response times should not delay users during customer interactions.
- All data that has been entered should be retrievable at any later date.

In order to realize specific business and system objectives, you often need to start by identifying higher-level objectives, continually refining until specific objectives are determined.

To determine objectives:**1. Identify high-level objectives.**

Objectives are often very general, and nonquantifiable. For example, Global Company might have the following high-level objectives:

- To allow customers to order products through a web site
- To reuse the existing system where possible
- To replace current manual processes with automated processes

2. Make the objectives more specific.

In order for objectives to be specific, they must be measurable, achievable, realistic, and timely. Additionally, by making the objectives measurable, it will be easy to determine if the completed project meets the objectives.

For example, Global Company's high-level objectives might be refined to these more specific objectives:

- To allow 80% of customers to create their own orders using a web site
- To reuse 50% of existing functionality
- To automate 50% of existing manual tasks

3. Determine system objectives.

Some business objectives will lead to system objectives. These objectives should be handled in the same manner as business objectives. That is, they should be specific, measurable, achievable, realistic, and timely.

4. Prioritize objectives.

Prioritizing ensures that when project development begins, the focus is on those items that help to achieve the objectives with the highest priorities. For example, you might use the following categories to help prioritize:

- Must have: Critical to the project's success
- Should have: Not critical, but should be considered mandatory
- Could have: Desirable, but the project will still succeed without them
- Won't have: Deferred goals for a future project due to constraints

3.2.3 Determining Business and Operational Requirements

To help determine business and operational requirements, you build a use case model. When building the model, focus on addressing the defined business objectives. In a use case, you define who or what (the actor) interacts with the system, and the goals of that interaction. The use case results in a business interaction diagram used in the discovery phase, where you determine the needed services to fulfill the use case, and further identify those services that exist and those that will need to be developed.

To create a use case model:

1. Determine the actors that will interact with the system.

Actors represent users and other systems that will interact with the project's system. For Global Company's Order Booking application, the following actors might be defined:

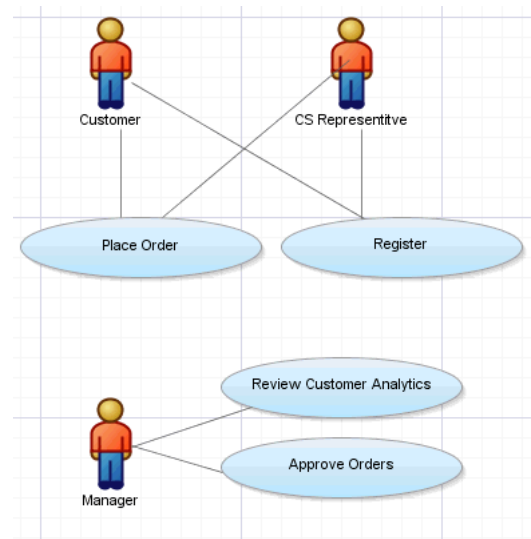
- Customer: The person registering, placing orders, and reviewing past orders on the web site
- Customer Service Representatives: The internal employee accessing the system to place and review orders on behalf of a customer
- Manager: The internal employee approving order
- CRM application: The existing application that stores customer information
- Credit verification system: The system that verifies customer credit information
- Rapid Manufacturer: The company supplying prices and products for orders
- Select Manufacturer: The Company supplying prices and products for orders

2. Define use case packages for each actor.

Use case packages are high-level use cases for each actor.

For example, [Figure 3-4](#) shows the high-level use cases for some of Global Company's actors. While the customer and customer service representative both need to place orders and register, the manager needs to review customer analytics and approve orders.

Figure 3-4 High-Level Use Cases in an SOA Project



3. Examine each use case package, and determine whether any can actually be a SOBA.

SOBAs are use case packages that encompass a specific business process. The Place Order package can actually be defined as two separate SOBAs. One would be for shopping for the products and creating the order, while the other might be the order fulfillment process.

For example, Global Company may determine that the following SOBAs are needed:

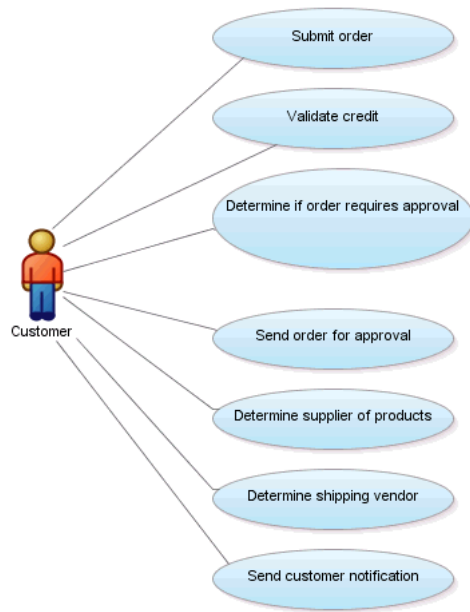
- Manage customers
- Shop for products and create order
- Book orders
- Review orders

In the next step, you'll decompose the high-level use case packages, or SOBAs, into more detailed use cases. This will allow you to begin to identify the services that will be exposed by each SOBA.

4. Define use cases for each SOBA.

The use cases provide a lower-level view of the steps for each SOBA. For example, the Order Booking SOBA might contain use cases for submitting an order, validating credit, approving an order, determining the supplier, and notifying the customer, as shown in [Figure 3-5](#):

Figure 3-5 Use Cases



5. For each use case, define any preconditions and post-condition guarantees.

Preconditions are requirements that must be satisfied before the use case can begin. Post-condition guarantees are what will happen when the use case is successfully completed.

For example, for the Validate Credit use case, a precondition might be that the customer's credit card type and number must be known. A post-condition guarantee might be that the external validation system returns a validation message.

6. For each use case, describe the main success scenario.

This scenario includes the steps needed to satisfy the use case. For example, the Validate Credit use case might have the following steps:

- a. Obtain credit card information.

Tip: At this point, you do not identify where this information comes from or how it will be retrieved.

- b. Send the card information out for validation.
- c. Receive validation from the credit company.

7. For each step in the scenario, identify failure conditions and extensions.

Failure conditions are places where something can happen that jeopardizes the success of the use case. Note that some failures can be avoided by determining whether the use case can follow an alternate path to success. These alternate paths are referred to as *extensions*. In an SOA project, exceptions and extensions need to take into account all actors (including secondary systems) involved in the use case.

For example, a failure condition might be identified for step 2 in the Credit Validation use case, for when the credit card does not pass validation.

8. Define the recovery steps for each failure condition.

These recovery steps should be treated as small use cases. That is, the steps should be a success scenario, with associated failure conditions noted. Continue in this fashion until all failure conditions are discovered.

For example, the recovery steps for a validation failure might be:

- a. Set the order status to canceled.
- b. Send an email to the customer explaining why the order was canceled.

The second step in this recovery scenario might also have a failure condition for when the email is returned undeliverable.

9. Add any supplementary requirements.

There may be additional requirements that are not fully surfaced in a use case. For example, reuse and automation are two business objectives for Global Company. Because of this, a supplemental requirement for the Credit Validation use case might be that the process of communicating with the credit company be fully automated, and that both the customer and the internal representative be able to use the same process.

Additionally, because there is now a supplemental requirement that the process be automated, you might add another supplemental system requirement regarding the communication process with the credit company.

10. Consider any deployment constraints.

This is especially important in an SOA project, as there are many actors that may be external to the enterprise.

For more detailed information about creating and working with use cases, refer to the following documents on the Oracle Technology Network:

- *Defining Business Requirements using UML Use Case Diagrams in JDeveloper 10g*
http://www.oracle.com/technology/obe/obe9051jdev/Modeling/UML_UseCase.htm
- *Design First, Code Next*
http://www.oracle.com/technology/pub/articles/masterj2ee/j2ee_wk3.html

3.2.4 Conducting a Discovery Phase

In the discovery phase, you first create a high-level design of the solution based on the use cases defined in the previous phase, without regard to any existing functionality. You then map existing services from the service portfolio to this high-level design. This mapping determines the services and functionality that must be created, that currently exist, and that exist but require some modification to satisfy the requirements.

You start this phase by conducting the discovery process on each of the SOBAs, as defined in [Section 3.2.3, "Determining Business and Operational Requirements"](#). For example, Global Company might conduct separate discovery phases for the Place Order and Register SOBAs. However, it is important that before any planning begins all dependencies between the packages be noted.

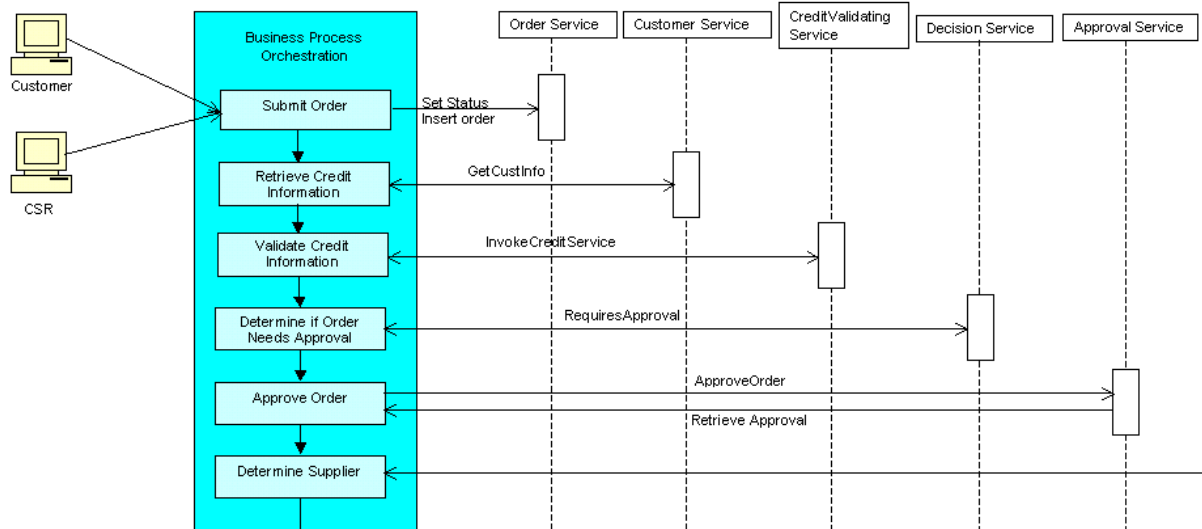
To conduct a discovery phase:

1. Define a process flow for each SOBA by creating a business interaction diagram.

When you begin to examine each SOBA defined in the previous phase, you may find that most of them are process-centric, meaning that they are basically the embodiment of high-level business processes that are orchestrated from some service-level endpoint. When this is the case, you should create a business interaction diagram for the SOBA. This diagram shows how each step in the process interacts with some service to achieve a goal, and how the services are orchestrated into the single process flow. It also shows whether the services should be invoked synchronously or whether they will require an asynchronous invocation.

For example, since Global Company knows that the order approval process will remain a manual task (the manager still must manually review and approve the order), the invocation will need to be asynchronous.

The Order Booking SOBA might have a business interaction diagram similar to [Figure 3–6](#). Each step in the process flow points to some service that allows the process to continue to the next step.

Figure 3–6 A Partial Business Interaction Diagram for the Order Booking SOBA

Notice that by creating this diagram, Global Company discovers that the Order Booking SOBA will require the following (partial) list of services:

- Order
- Customer
- Credit Validation
- Decision
- Approval

Tip: The Customer service in this SOBA is likely the same as the Customer service in the Registration SOBA. For this reason, it is important when defining and designing services that you keep in mind all reuse between SOBAs throughout the project.

2. Define the discovered services.

Once all services are discovered, you can begin to define each service. The definition should remain at a high level, as the details will be determined during the design phase. For each service, you should define the following:

- Service name: A unique name for the service.
- Service description: An overview of the service.
- Service type: Synchronous or asynchronous.
- Functional domain: The business domain the service will belong to, as defined during the domain decomposition of the service portfolio planning.

- Service classification: The service can be one of the following classifications:
 - Component service: A simple atomic service potentially acting on a single enterprise resource. Examples are database and code.
 - Data service: A service that provides data querying and/or combination and transformation for multiple data sources.
 - Business service: An atomic service composed of combinations of component services and business rules, for example, aggregator services or business delegates.
 - Workflow service: A long-lived business process that coordinates with other services and has external interactions.
- Service process rules: Details of the service processing logic.
- Message formats: Inbound and outbound messaging formats, including details of any fault message format for the synchronous services.
- Security requirements: Security in relation to client authentication, authorization, and data encryption.
- Applications: The data sources (for example, an application or database) this service interacts with to complete its functionality, including the details of the communication interface if available.
- Service dependency: Any other services this service depends on for functionality.
- Human workflow: Any human workflow capabilities.
- Performance goals and measures: Specific performance metrics for the service. These can include service processing time, concurrent users, message data sizes, and so on.
- Exception handling: Any exceptions and exception handling logic, including criteria for when faults are generated by the service.

For example, the `CreditValidatingService` definition in the Order Booking SOBA might be as shown in [Table 3–1](#).

Table 3–1 *CreditValidatingService Service Definition*

Item	Description
Service Name	<code>CreditValidatingService</code>
Service Description	Determines if a customer's credit card is valid
Service Type	Synchronous
Functional Domain	Order Fulfillment
Service Classification	Business Service
Service Process Rules	A message with credit card type and number is sent to the external credit validation application. This application returns <code>true</code> if the card is valid, <code>false</code> if it is not.
Message Formats	XML
Security Requirements	The credit card number must be encrypted.
Applications	Interfaces with an external credit validation system. This system will provide a <code>wsdl</code> file to which all messages must adhere.
Service Dependency	The <code>Customer</code> service must provide the credit card information.
Human Workflow	None
Performance Goals and Measures	Must be available 24/7
Exception Handling	A fault is invoked when the credit card is not valid. This should set the status of the order to <code>cancelled</code> .

3. Compare each of the defined services with the service portfolio created in [3.2.1](#) , "[Creating a Service Portfolio Plan](#)".

You may find that a service required by the project already exists in the portfolio, either as an implemented service or as a conceptual service. If the project requires a service that is not part of the portfolio (either as an implemented service or as a conceptual service), it should be added. If a service exists as a conceptual service in the portfolio, then it needs to be created as part of the project.

3.2.5 Defining the Project

Once you have determined the business flows and services that need to be implemented, you further define how the flows will orchestrate the services. In this phase, you determine how the services will work together by defining the business process model logic, process variable definitions, and more detailed exception handling. For services, you define the responsibilities of the services, along with their needed granularity.

To define the orchestration:

1. Analyze the use cases, and determine how they will all work together.

For example, during the requirements and discovery phases, it was determined that Global Company will need the following SOBAs:

- **Customer Registration:** While customer registration functionality exists for internal customer service representatives, Global Company needs to implement a web-based registration process so that customers can register themselves.
- **Create Order:** Similar to the registration process, Global Company needs to implement a web-based shopping site for their customers.
- **Order Booking:** Although this business process currently exists for internal customer service representatives, Global Company's requirements state that the manual processes should be automated, and then used for both the internal employees and external customers. This means that the process and services must be accessible from both a web application and the internal application.
- **Review Orders:** This is an existing process where a manager must manually review an order to determine if the order should be fulfilled. This service must also be available to both the web site and the internal customer representative system.

Since the registration process will interface with the existing CRM application, Global company decides that the Customer service discovered in the last phase will interface directly with the CRM application. The flow of the registration process is defined for the web site.

Similarly, the web site will interface directly with the product database to allow customers to browse and select products. This flow is also defined. Both the registration and shopping process are simple flows. Because of this, Global Company determines they will not require a BPEL process flow.

Further definition and design of the web site can follow any standard web design process, including the definition of any needed classes to support the flow of information within the web site. For example, an entity class that will handle the display of products for the web site needs to be defined.

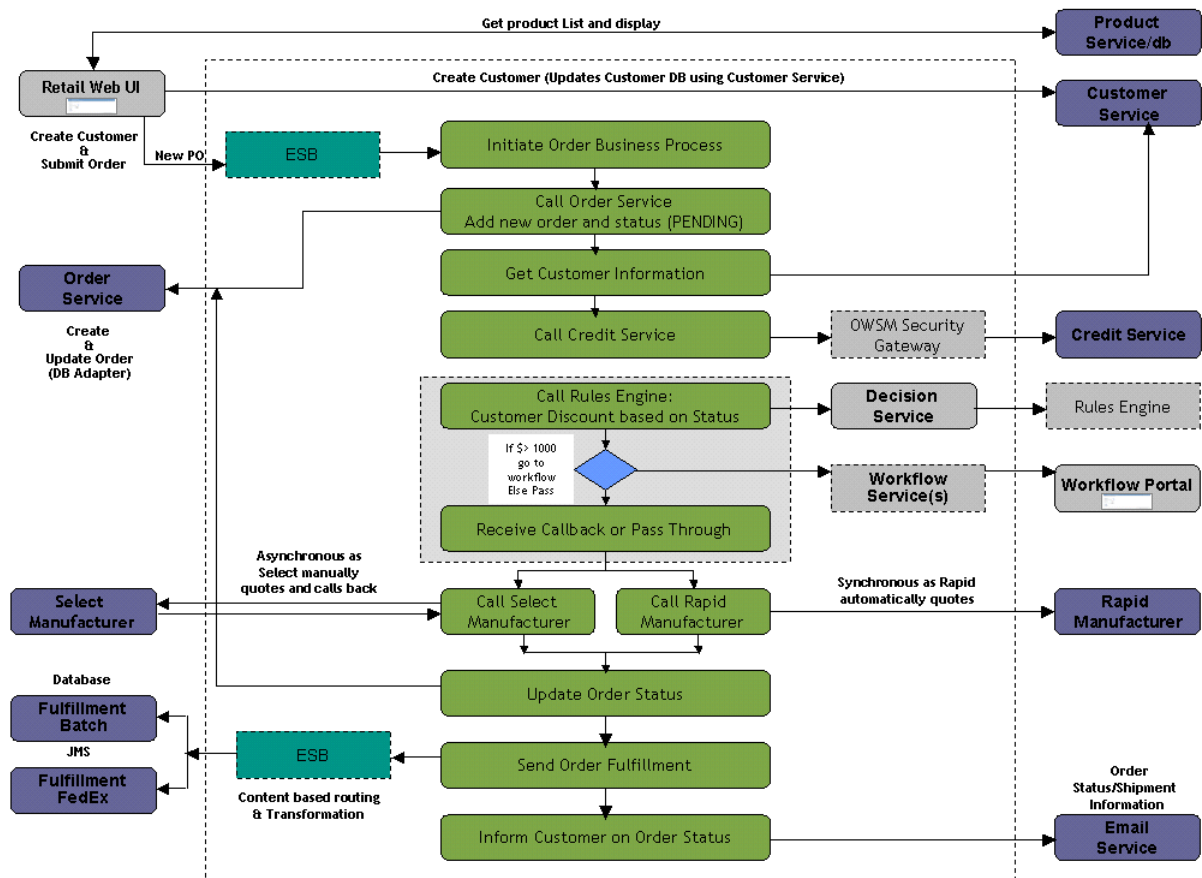
Since the order booking and order approval functionality needs to be accessed by both the internal customer service application and the customer web application, Global Company decides to use an enterprise service bus (ESB) to invoke that process. And since that process is long running and orchestrates many different services, they decide to use a BPEL process flow to manage the business process.

Because the order approval process needs to remain a manual workflow, they decide to keep that flow as is, but to automate its invocation.

2. Define the different process flows and ESBs.

Once you've determined where BPEL flows can be used, you can begin to define how they will orchestrate the services. [Figure 3-7](#) shows how Global Company orchestrates the registration and order booking processes. Note how the focus in this diagram is more on how the system will work, rather than on how the system accomplishes a business goal.

Figure 3-7 A High-Level Design for Global Company.



For each process flow and ESB, define the following:

- **Flow logic:** Define what the flow will do, the steps it will need to include, and the way in which it will interact with the services.

For example, the ESB used to route fulfillment information will need to transform order information to data that can be consumed by a batch process used by the USPS, and also by a database adapter that will populate the data into a table.

- **Subprocesses:** Define any subprocesses for the flow, at the same level of detail.
- **Variable definitions:** Define variables that will be needed to hold information used throughout the flow.
- **Business process flow scopes:** Define the scopes for the flow. A scope groups a collection of nested activities that are centered around a particular service, and contains a primary activity that defines its behavior. Scopes can have their own local variables, fault handlers, and so on.

For example, the OrderBooking flow has the CreditService scope, which contains all the activities necessary to validate credit information.

- **Exception handling:** Define all possible errors that can occur in the flow, and how, technically, they should be handled. For example, determine how time-outs will be handled.

3. Analyze, define, and classify the services.

You define services in much the same way as analyzing and defining classes in a standard application. For example, in a typical class analysis phase, you might identify analysis classes, and classify them into boundary /interface classes, entity classes, and control classes.

However, when you define a service for an SOA system, your focus should be on the service's responsibilities to other components in the system. Additionally, you must define the granularity of the service. The following are important aspects in determining granularity:

- **Business suitability:** Ideally, a service supports an atomic operation of a business task. It provides just enough functionality for the given task. Making the service more simple makes it more general, and therefore, increases the likelihood of reuse.

For example, the Order Booking flow contains both an `Order` and an `Order Status` service. The `Order` service is used just to persist order information. However, because the status of an order needs to be updated at various points during the flow, the updating status functionality was extracted. This extraction allowed that discreet functionality to be reused.

- **Performance and size:** Because services are called remotely (and therefore require a round trip), they incur performance overhead. To avoid this overhead, you want to call a service as few times as possible. Doing so needs to be balanced with the service having the right level of functionality (as described in the previous point). For example, you don't want to make services so granular that you would need many to make frequent calls to many services to complete a single task.

For example, instead of defining one service that handles orders, and another that handles individual order items in an order, the Order Booking system has one service that handles both orders and order items.

Additionally, services are limited by the size of the messages they can efficiently process. When a message becomes too big, it will require too many resources to process.

- **State management:** Transactions and state operations should be self-contained whenever possible. Doing so reduces the risk that the state will be lost if a service fails to complete. Whenever a service stores data, it should be done in one transaction, and failure of the transaction must not result in lost or corrupt data.
- **Location transparency:** All services should be invoked remotely. There should be no dependencies on where the service is located.
- **Implementation independence:** All services should use contract-based design, meaning that the interface they implement should remain stable and easily extendable. The implementation behind the interface should be able to change easily, without needing to change the interface.

Now that you have completed the discovery phase, and know more about how the services need to be created and used, you can further classify each service.

Typical service classifications are as follows:

- Functionality provided by the service
 - Infrastructure, for example, a DNS lookup
 - Data, for example, a federated query
 - Business logic, for example, a fraud-check algorithm
 - Utility, for example, transformation or routing
 - Information system, for example, an ERP function
 - Process control, for example, an approval process
 - UI, for example, a portlet
- How the service is used
 - High-level business, for example, PO processing
 - Supporting business, for example, a PO approval workflow
 - High-level technology, for example, a user lookup
 - Supporting technology, for example, logging
- How the service is constructed
 - Simple, for example, a web service provided by an out-of-the-box email application
 - Wrapped, for example, a web service invoking functionality in an existing CRM application.
 - Composite, for example, a service that combines bids from multiple suppliers.
- How the service is invoked
 - Synchronous/Asynchronous, for example, a request/reply to specific service
 - Event-based, for example, notifications for any interested party to consume

Services typically belong to more than one classification. For example, the `CreditValidation` service, first described in [Table 3-1](#), might now be further classified as a utility service that provides supporting business logic, is a wrapper to an existing application, and is invoked synchronously.

This type of classification allows you to more easily adhere to any existing best practices for specific classifications. For example, Global Company may have standards for data access and security for any services that are classified as data services. Additionally, Global Company may have a standard that states that all services classified as composite services should use a BPEL flow to orchestrate the composition, and that those flows must adhere to existing standards. Using the classifications to help drive best practices and standards is further discussed in [Section 3.3.2, "Designing Services"](#).

4. Identify common, typically nonfunctional requirements for each flow and service.
Review any nonfunctional requirements described in the use cases and translate them to specific requirements. These requirements will drive the architectural design process. Some typical operational requirements include:
 - Throughput
 - Response time
 - Round-trip
 - Latency
 - High availability
 - Fault tolerance
 - Security
 - Access control
 - Encryption
 - Monitoring
 - Alerts
 - Key performance indicators

3.2.6 Creating a Solution Specification

At this point, it is useful to develop a solution specification. A solution specification provides a mechanism by which stakeholders can review the proposed system. Doing so ensures that the solution realizes the initial business objectives.

While companies will have their own requirements for a solution specification, the following topics must be covered for an SOA solution:

- Background
 - Important enterprise-level objectives regarding SOA
 - Key governance principles to which SOA systems must adhere
- Project scope
 - Business goals and objectives as defined in [Section 3.2.2, "Determining Business and System Objectives"](#)
 - Business requirements as developed in [Section 3.2.3, "Determining Business and Operational Requirements"](#)
- SOBAs
 - Required service and process portfolios as defined in [Section 3.2.5, "Defining the Project"](#)
- High-level architecture
 - Component view as the services are described in [Section 3.2.5, "Defining the Project"](#)
 - Process flow view as also described in [Section 3.2.5, "Defining the Project"](#)

- Operational requirements
 - Performance and other nonfunctional requirements discovered documented during the various design phases
 - Availability, Security and Manageability requirements
- High-level synopsis of standards and project delivery best practices (detailed standards will be documented during the design phase)
- SOA governance compliance check to ensure that the solution specification not only satisfies functional and operational requirements, but that it does so in a way so as to comply with all applicable architecture, design, implementation, and regulatory best practices of the company.

3.3 Designing the Project in the Application Scope

You complete the design process within the application scope. Here you create technical designs for each of the constructs (that is, business flows, ESBs, services, and so on.), discovered and defined in the project scope. You will decompose these constructs into finer-grained constructs, until the design is such that you can determine how best to use the products within Oracle SOA Suite to create runtime executables.

3.3.1 Designing a Process Flow

You can now add implementation details for process flows, using the process flow definitions created in [Section 3.2.5, "Defining the Project"](#).

To design a process flow:

1. Using the use case analysis and the business interaction diagram, design the activities for each step within each scope of a flow.

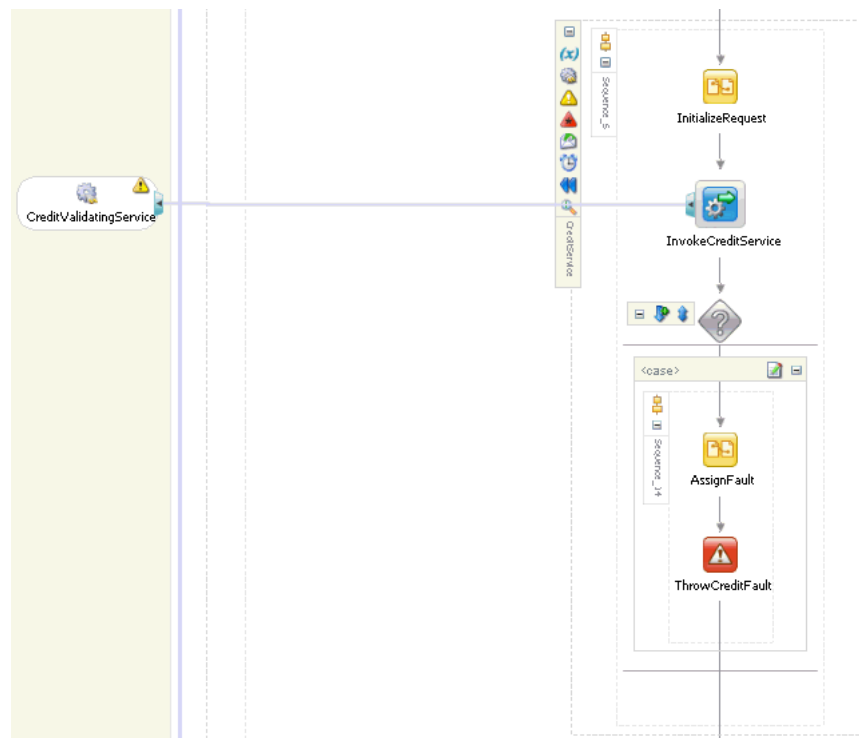
For example, based on the definition created in [Section 3.2.5, "Defining the Project"](#), you might start by designing each scope within a flow. You might then begin to add all activities that need to be completed for that scope, in order to move on to the next scope. Types of activities include:

- Assign: Copies data from one XML variable to another
- Invoke: Sends information, typically a request, to a web service, which is identified by its partner link, and specifies an operation for the web service to perform
- Partner Link: Characterizes the conversational relationship between two services by defining the roles played by each service in the conversation and specifying the port type provided by each service
- Receive: Waits for a message from the client or a callback response message from an asynchronous web service
- Reply: Sends a message in response to an invocation

For example, [Figure 3–8](#) shows the activities for the `CreditValidation` scope, which include:

- `InitializeRequest`: Copies the credit card type and number from the customer data and assigns the values to variables.
- `InvokeCreditService`: Invokes the `CreditValidatingService` web service with a synchronous call that includes the credit card number and type. The service replies with either a valid or not valid response.
- Generic switch activity: An expression in the switch task states that if the service returns a response stating that the card is not valid, an error must be thrown. The switch contains the following two activities:
 - `AssignFault`: Assigns the value "Credit Problem" to the status variable.
 - `ThrowCreditFault`: Invokes the `OrderBookingFault` scope.

Figure 3–8 The Credit Validation Scope



2. For each activity (including the scope activities), define the following:
 - Activity name: Name of the activity.
 - Business process: The process flow to which the activity belongs.
 - Activity description: An overview of the functionality the activity provides.
 - Activity type: Web service, WSDL EJB Binding, Java code, Transformation, Assignment, and so on.
 - Activity details: Detailed description of the functionality provided by the activity.

- Transformation details (when applicable): Data mapping details. See step 3 for more information.
- Parameters: Process parameters modified or used by this activity.
- Exceptions: Any faults that can be generated by the activity. Include exception handling logic.

[Table 3–2](#) shows the definition for the activity that invokes the `CreditValidating` service to verify a customer’s credit card.

Table 3–2 InvokeCreditService Activity Design Description

Item	Description
Activity Name	<code>InvokeCreditService</code>
Business Process	<code>SOAOrderBooking</code>
Activity Description	Invokes the <code>Credit</code> service to verify a customer’s credit card
Activity Type	Web service
Activity Details	Synchronously invokes the <code>Credit</code> service and sends credit card type and number. Waits for a response that contains either an <code>valid=true</code> or <code>valid=false</code> message.
Transformation Details	None
Parameters	<code>ccType</code> , <code>ccNum</code>
Exceptions	The <code>Credit</code> service may not be available, or may not send a response. If unable to connect to the service, throw a <code>serviceNotAvailable</code> exception. Need to set a time-out that will be used to determine when to throw a <code>noResponse</code> exception. If the credit card is not valid, throw a fault that will set the order status to <code>canceled</code>. Because there are other places in the flow where an order should be set to <code>canceled</code>, this should be its own subflow that can then be reused.

3. Provide details of any needed document data transformation.

Typically, data transformation converts inbound data to variables previously defined for the process, or it converts a variable to outbound data structures required by the destination system. These variables represent canonical business objects. For example, the `TransformOrder` activity in the `SOAOrderBooking` flow transforms order information received from the ESB (`inputVariable` variable) to the `orderRequest` variable, which will be used to persist order information into the database. You create a data transformation specification to provide detailed node-to-node mappings, as shown in [Table 3–3](#):

Table 3–3 Data Transformation Mapping Requirements

Item	Description
Source Node Name	Node name in source object.
Source Node Path	Path to node within source object.
Source Node Description	Description of the node.
Source Data Format Definition	Data type. For example, <code>char</code> , <code>integer</code> , <code>date/time</code> , <code>binary</code> encoded. Also note the maximum length.
Destination Node Name	Node name in the destination object.

Table 3–3 (Cont.) Data Transformation Mapping Requirements

Item	Description
Destination Node Path	Path to the node within the destination object.
Destination Node Description	Description of the node.
Destination Data Format Definition	Data type. For example, char, integer, date/time, binary encoded. Also note the maximum length.
Mapping Type	Straight: No data conversion. Derived: Derived from multiple inbound nodes. Complex: Complex mapping logic; database/table lookups.
Mapping Details	Detailed description of mapping logic.

4. If the activity invokes an external service, you should note the following for that trading partner:
 - Exchange protocols
 - Document protocols
 - Message types
 - Constants for selected protocols
 - Endpoints (for example, https, email, ftp)
 - Design for SLA failures
 - Contact information
5. If the process flow will be invoked as a web service, determine any authentication or authorization needs for the flow.

For example, you may want to restrict access to a flow based on user type, or you may need to encrypt data.

3.3.2 Designing Services

To create a technical design for a service, you start with the service definition created in [Section 3.2.5, "Defining the Project"](#), and then add detailed technical elements.

To design a service:

1. For each defined service, create a technical design that details the following items:
 - Message formats: XML definitions for inbound and outbound objects.
 - Business process rules: Component design to meet business process rules.
 - Data sources: Detailed sequence diagrams for interacting with application APIs, EJB methods, Java class methods, and so on.
 - Process Security: Configuration details for service security, for example, AppService JAAS configuration or WS-Security usage.
 - Exception handling: Detailed error processing options and components necessary for error handling. Possible design options include error trapping, editing, and replay capabilities and detailed service fault definitions.

- Human Workflow: Design of the human workflow system and interaction with business process (if applicable). Includes user interface design and page flows.
2. Assign design patterns to each of the defined services, using their classifications created in [Section 3.2.5, "Defining the Project"](#) as a guide.

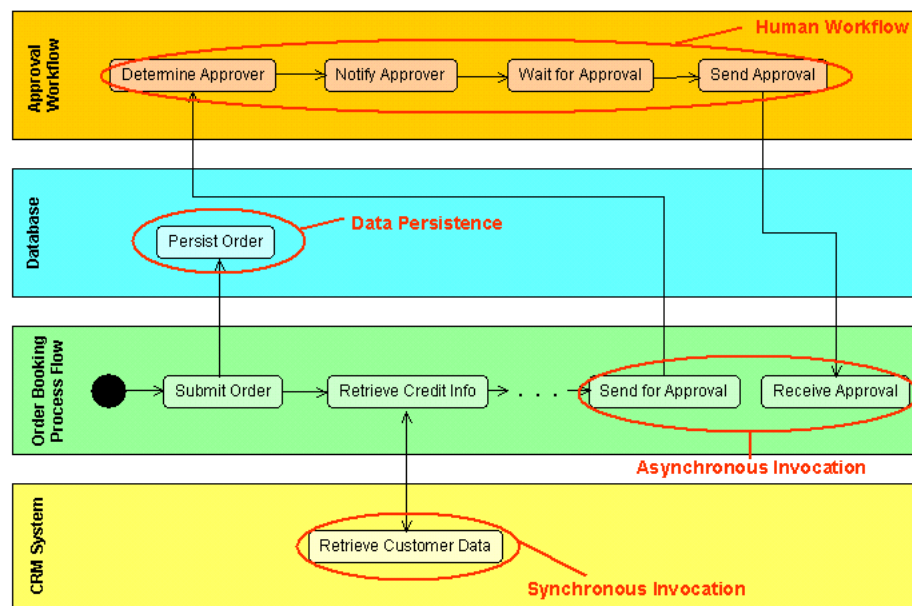
By assigning design patterns to particular classifications, you ensure that coding best-practices are adhered to throughout the system.

For example, note the classifications of the following services:

- Order: Data functionality
- Customer: Wrapper on an information system invoked synchronously
- ApproveOrder: Human workflow invoked asynchronously

Figure 3–9 shows how these categorizations can lead to assigning designing patterns (shown in red). All other services falling into the same categorizations will be assigned the same design patterns, and will therefore follow set standards and guidelines for those patterns.

Figure 3–9 Service Categorization Leads to Pattern Assignments



Examples of integration design patterns include:

- Facade (also known as *Wrapper*)
- VETRO (Value-base, Enrich, Transform, Route, Operation)
- EAI
- Human Workflow
- Hybrid (a mix of automatic and human workflow)
- Service Broker
- Rules
- Invocation

3. Design the service's interface.

This design can follow any standard component design process. However, you should review designs to ensure that dependencies between services are minimal, and that any dependencies are written in the contract (the interface), and not the component itself. This interface is what will become the web service used by the system.

For example, the `CreditValidationService` requires information that is supplied by the `Customer` service. Instead of interacting with the `Customer` service directly, activities are added to the flow to access the needed data and store it in variables. The `CreditValidationService` then uses those variable values to perform the validation. In the future, either of these loosely coupled service implementations can change without affecting the interaction between them.

4. Ensure that all original business and operational requirements are met by the service.

For example, if the `Order Booking SOBA` has a specific transactional requirement, then every service endpoint that is part of that process must meet that requirement. If they cannot, then certain compensation patterns may be applied to provide the needed transactional integrity.

3.3.3 Creating a Physical Design

Now that you have determined exactly what needs to be created, you can determine how to implement each of the flows, ESBs, services, and service components using Oracle SOA Suite.

To create a physical design:

1. Map each construct from the designs to a product in Oracle SOA Suite that can be used to develop the construct. This suite provides a complete set of service infrastructure components for creating, deploying, and managing SOA-compliant applications and processes.

Oracle SOA Suite offers the following applications to help you in implementing an SOA system:

- **Oracle JDeveloper:** Provides an ISE (integrated service environment) to complement its IDE functionality. Visual and declarative tooling for BPEL design, ESB system design, and XSLT mapping are an integral part of JDeveloper. JDeveloper also supports the complete development life cycle with integrated features for modeling, coding, debugging, testing, and tuning applications.

For information about using JDeveloper to create EJB service components, see [Chapter 4, "Building and Using Application Services"](#). For information about using JDeveloper to create and manage web services, see [Chapter 5, "Building Web Services"](#). For information about developing a web client using JDeveloper, see [Chapter 9, "Developing a Web Application in an SOA System"](#). JDeveloper is also used to design and develop the ESB and BPL flow. See the following points for more information.

- Oracle Business Rules: Provides the infrastructure for the development and deployment of business rules. Oracle Business Rules consists of a Rule Authoring Tool for defining rules, an SDK that provides rules access and update for embedded programs, and a Rules engine that executes the rules. For more information about creating and deploying rules using Oracle Business Rules, see [Chapter 8, "Creating and Using Rules for a Decision Service"](#).
 - Oracle Enterprise Service Bus: Moves data among multiple endpoints, both within and outside of an enterprise. Oracle Enterprise Service Bus uses open standards to connect, transform, and route business documents as XML messages among disparate applications. You design an Oracle ESB using the ESB Designer within JDeveloper. For more information about creating ESB systems, see [Chapter 6, "Using Oracle Enterprise Service Bus"](#).
 - Oracle BPEL Process Manager: Offers a comprehensive and easy-to-use infrastructure for creating, deploying and managing BPEL business processes. You develop BPEL process flows using the BPEL Designer within JDeveloper. For more information about creating process flows, see [Chapter 7, "Using Oracle BPEL Process Manager"](#).
 - Oracle Web Services Manager: Allows you to centrally define policies that govern web services operations (such as access policy, logging policy, and load balancing), and then wrap these policies around web services without requiring modification to those services. For more information about creating policies, see [Chapter 10, "Securing the System"](#).
2. Based on the chosen tools, and the assigned design patterns, you can now develop best practices and standards for implementation.

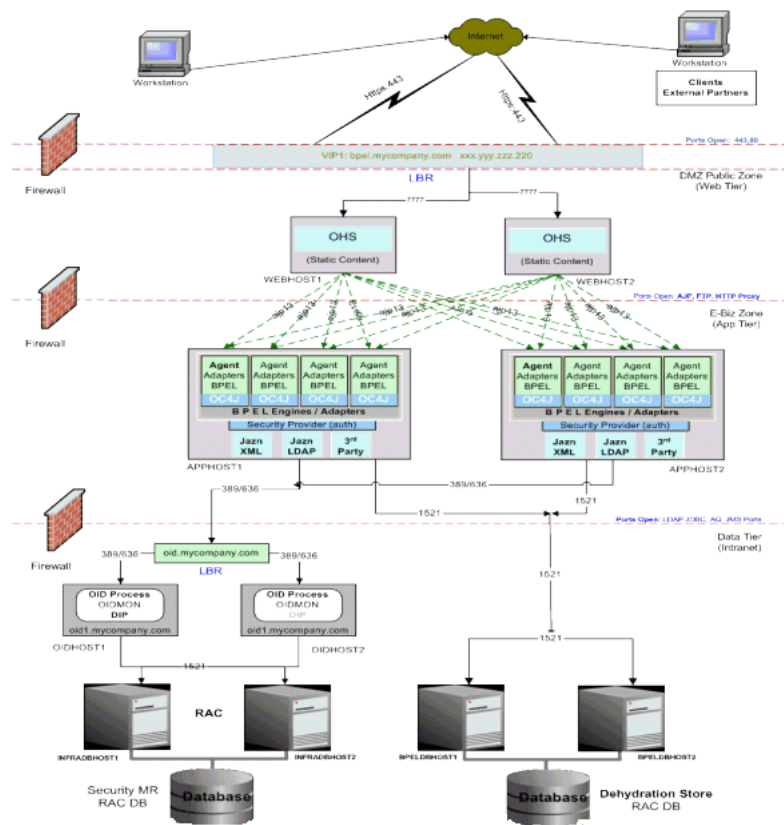
In an SOA system, it is important that naming conventions be defined for all project components, and that they be tailored to the different products used for implementation. For example, service names, service operation names, BPEL process names, and BPEL activity names, should all be standardized.

Coding standards for an SOA system should be addressed for each of the employed design patterns. Some examples include:

- Common WSDL design standards, such as standard fault replies for synchronous web services and business processes.
 - Common error handling approach
 - Common logging approach
3. Create a deployment topology.

This topology maps each application component to the system architecture. It should identify the specific software, servers, server configurations, and io subsystem configuration. [Figure 3–10](#) shows an example of a deployment topology for a BPEL process flow.

Figure 3–10 Deployment Topology for a Process Flow



Building and Using Application Services

This chapter describes how to develop the EJB service portion of an application using JPA (EJB 3.0) entities to persist data, an EJB session bean to implement a session facade, and how to expose the functionality as a web service and an ADF Data Control.

Note: The application services described here are used in the Customer Service application in the SOA Order Booking sample application. Architects and developers are not confined to using this technology stack and should not feel limited to this choice for their own applications.

This chapter includes the following sections:

- [Section 4.1, "Introduction to Business Services"](#)
- [Section 4.2, "Creating JPA Entities that Map to Database Tables"](#)
- [Section 4.3, "Configuring Persistence Units"](#)
- [Section 4.4, "Implementing Services with EJB Session Beans"](#)
- [Section 4.5, "Accessing Entities from a Session Bean"](#)
- [Section 4.6, "Exposing a Session Bean as a Web Service"](#)
- [Section 4.7, "Exposing Services with ADF Data Controls"](#)
- [Section 4.8, "Testing EJBs"](#)
- [Section 4.9, "Deploying EJB Applications"](#)

4.1 Introduction to Business Services

When you create an EJB project, you can start by creating either the services (session beans) or the persistence (JPA entities). Often a developer may create the service and a set of business methods first, and implement the persistence portion later. For applications where you are only using the facade-generated methods, it may make more sense to create the persistence units first. In the case of a small application, like the Customer Service portion of the Order Booking demo application, creating the persistence units first requires fewer steps, and that is how the process is described here.

Tip: Most teams have their own respective source control management (SCM) procedures, policies, and common philosophies towards what constitutes a transaction or unit of work for the SCM system. In the absence of a policy, you should group logical changes into a transaction, and also commit your changes when you need to share your modifications with another member of your team. In general, it is not advisable to commit changes when they do not compile cleanly or pass the unit test created for them.

4.2 Creating JPA Entities that Map to Database Tables

The easiest way to create JPA entities that map to database tables is to use the Create Entities from Tables wizard. The wizard guides you through the process of obtaining or creating a database connection, choosing the tables you want to use, and creating JPA entities.

The Create Entities from Tables wizard is available in the New Gallery, in the Business Tier section.

Tip: When you create entities from tables, you can create more than one at the same time. If the tables in the database are related by foreign key constraints, fields will be created and annotated in the generated entities. For this reason, it's often a best practice to generate all your entities at the same time.

4.2.1 What Happens When You Create JPA Entities From Database Tables

When you create JPA entities from tables, new Java classes are created to support each entity. If there are relationships between the database tables, these are discovered and preserved via annotations in the source code.

In [Example 4-3](#), notice the annotations in the source code that begin with the `@` symbol. Annotations vastly simplify working with EJB entities.

Example 4-1 Customer Entity Created From a Database Table

```
package org.soademo.customerservice.persistence;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;
import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.JoinTable;
import javax.persistence.NamedQueries;
import javax.persistence.NamedQuery;
import javax.persistence.OneToOne;
import javax.persistence.SequenceGenerator;
import javax.persistence.Table;
import javax.persistence.TableGenerator;
```

```

@Entity
@NamedQueries( { @NamedQuery(name = "Customer.findAllCustomer",
                             query = "select object(o) from Customer o")
               ,
               @NamedQuery(name = "Customer.findCustomerById", query = "select
object(cust) from Customer cust where cust.custid = :custid")
               ,
               @NamedQuery(name = "Customer.findCustomerByEmail", query = "select
object(cust) from Customer cust where cust.email = :email and cust.password =
:password")
             } )

@Table(name = "CUSTOMER")
@SequenceGenerator(name = "SEQ_ID_GENERATOR", sequenceName = "EJB_SEQ_ID_GEN")
@TableGenerator(name = "TABLE_ID_GENERATOR", table = "EJB_TAB_ID_GEN",
                pkColumnName = "ID_NAME", valueColumnName = "SEQ_VALUE",
                pkColumnValue = "SEQ_GEN")
public class Customer implements Serializable {
    private String creditcardnumber;
    private String creditcardtype;
    @Id
    @GeneratedValue(strategy = GenerationType.TABLE,
                    generator = "TABLE_ID_GENERATOR")
    @Column(nullable = false)
    private String custid;
    private String email;
    private String fname;
    private String lname;
    private String phonenumber;
    private String status;
    private String password;

    @OneToMany(fetch = FetchType.LAZY, cascade = { CascadeType.ALL } )
    @JoinTable(name = "CUSTOMER_ADDRESS",
                joinColumns = { @JoinColumn(name = "CUSTID")
                              } , inverseJoinColumns = { @JoinColumn(name = "ADDRESSID")
                              } )
    private List<Address> addressList;

    public Customer() {
    }

    public String getCreditcardnumber() {
        return creditcardnumber;
    }

    public void setCreditcardnumber(String creditcardnumber) {
        this.creditcardnumber = creditcardnumber;
    }

    public String getCreditcardtype() {
        return creditcardtype;
    }

    public void setCreditcardtype(String creditcardtype) {
        this.creditcardtype = creditcardtype;
    }
}

```

```

public String getCustid() {
    return custid;
}

public void setCustid(String custid) {
    this.custid = custid;
}

public String getEmail() {
    return email;
}

public void setEmail(String email) {
    this.email = email;
}

public String getFname() {
    return fname;
}

public void setFname(String fname) {
    this.fname = fname;
}

public String getLname() {
    return lname;
}

public void setLname(String lname) {
    this.lname = lname;
}

public String getPhonenumber() {
    return phonenumber;
}

public void setPhonenumber(String phonenumber) {
    this.phonenumber = phonenumber;
}

public String getStatus() {
    return status;
}

public void setStatus(String status) {
    this.status = status;
}

public List<Address> getAddressList() {
    if (addressList == null) {
        addressList = new ArrayList();
    }
    return addressList;
}

public void setAddressList(List<Address> addressList) {
    this.addressList = addressList;
}

```

```

public Address addAddress(Address address) {
    getAddressList().add(address);
    return address;
}

public Address removeAddress(Address address) {
    getAddressList().remove(address);
    return address;
}

public void setPassword(String password) {
    this.password = password;
}

public String getPassword() {
    return password;
}
}

```

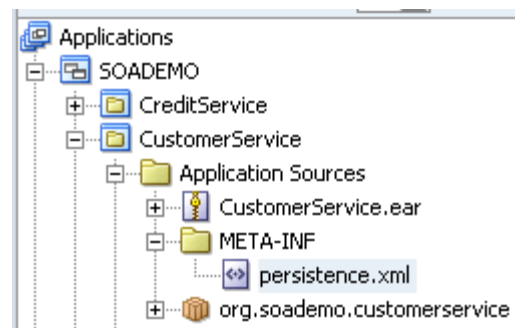
4.2.2 What You May Need to Know About Creating Entities From Tables

After creating an EJB entity from a database table, you can modify the generated annotations by hand. This is an advanced topic and not covered in this chapter.

4.3 Configuring Persistence Units

The persistence.xml file is a descriptor that declares that a .jar file is a persistence archive file containing JPA entities. The persistence.xml file is located in each entity project, in the Application Sources folder within the META-INF directory, as shown in [Figure 4-1](#):

Figure 4-1 Location of the persistence.xml file in its project



Within this file, there are one or more <persistence-unit> entries defined. Each persistence-unit identifies a subset (possibly all) of the entities included in the deployed JAR file, which may be a dedicated persistence archive, an EJB JAR file, or a WAR file. The persistence unit associates metadata with this entity group, including a named entity manager. The entity manager is covered in [Section 4.5, "Accessing Entities from a Session Bean"](#).

The same entity can appear in multiple persistence units, but hold different mapping behavior in each one. In addition to defining the entity classes that it represents, the persistence-unit defines its datasource information (when overriding the default connection) and any of a number of platform-specific properties. A common need for creating separate persistence units is to run the same entities both inside and outside the Java EE container. Testing EJBs outside the container is covered in [Section 4.8.1, "Testing Entities Outside the Container with a Java Service Facade"](#).

4.3.1 How to Create a Persistence Unit

To create a persistence unit, use the Create Persistence Unit dialog.

The Create Persistence Unit dialog lets you create persistence-units that are configured to run either inside or outside a Java EE container. When running inside a Java EE container, the unit can be associated with a JTA (Java Transaction API) datasource, which lets the entities participate in distributed transactions. The `<jta-data-source>` must be defined in the server. This may be defined in a `data-sources.xml` file provided during deployment, or deployed with the persistence unit, or it could have been pre-configured on the server. When running outside the container, entities can't participate in JTA transactions, so the properties that identify the datasource connection are defined explicitly.

To create a a persistence unit:

1. In the Navigator, right click the `persistence.xml` file and choose **New Persistence Unit**.
2. Enter the appropriate information in the Create Persistence Unit dialog. Click **Help** in the dialog or press the F1 key if you need more information.
3. Click **OK** to save your changes.

4.3.2 How to Edit a Persistence Unit

In JDeveloper 10.1.3.1, the easiest way to modify a persistence unit is to browse it in the Structure pane and modify it in the property inspector. However, if you want to edit an existing persistence unit by hand, you may do so by editing the XML code.

In the Navigator, double click the `persistence.xml` file to open it in the editor. Use the `persistence.xml` file from the SOA Order Booking application shown in [Example 4–2](#) as sample code.

Example 4–2 *persistence.xml* file

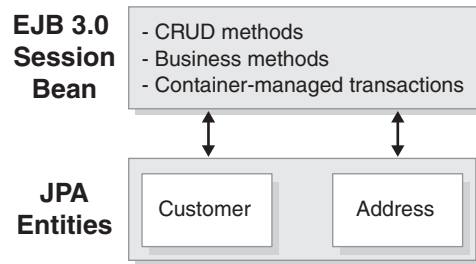
```
<?xml version="1.0" encoding="windows-1252" ?>
<persistence xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd"
             xmlns="http://java.sun.com/xml/ns/persistence">
  <persistence-unit name="customerServiceUnit">
    <provider>oracle.toplink.essentials.ejb.cmp3.EntityManagerFactoryProvider</provide
r>
    <jta-data-source>jdbc/soademoDS</jta-data-source>
  </persistence-unit>
</persistence>
```


4.4 Implementing Services with EJB Session Beans

A session bean exposes the functionality of the business layer to the client.

One common use of a session bean is to implement the session facade J2EE design pattern. A session facade is a session bean that aggregates data and presents it to the application through the service layer. Session facades have methods that access entities as well as methods that expose services to clients. Session beans have a transactional context via the container, so they automatically support persistent CRUD (Create, Retrieve, Update, Delete) functionality.

Figure 4–2 Session Facade Functionality



4.4.1 How to Create a Session Bean

To create a session bean, use the Create Session Bean wizard. This wizard is available from the New Gallery, in the Business Tier category.

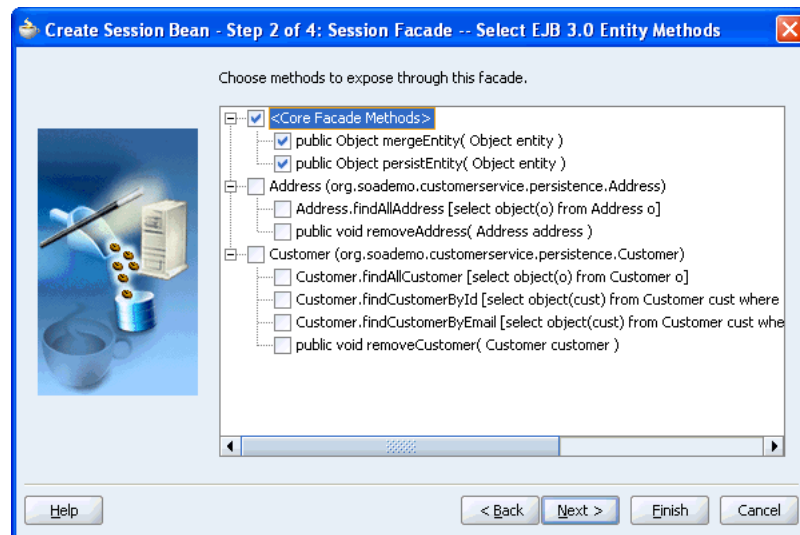
To create a session bean:

1. From the File menu in JDeveloper select New->Business Tier->EJB->Session Bean.
2. Follow the steps in the wizard to specify the EJB version, stateful and stateless functionality, and session facade methods. For more information on session facade methods, see [Section 4.4.1.1, "Generating Session Facade Methods"](#)
3. On the Class Definitions page, in the Bean Class field, choose a package. For more information about packages, see [Section 4.4.1.2, "Choosing a Package"](#).
4. On the Component Interface page, specify the interfaces you want to generate. For more information about interfaces, see [Section 4.4.1.3, "Remote, Local and Web Service Interfaces"](#).

Tip: For an SOA application, you should choose an EJB 3.0 version session bean, a stateless session, and container-managed transactions (CMT); bean-managed transactions (BMT) are beyond the scope of this book.

4.4.1.1 Generating Session Facade Methods

A session facade contains core CRUD methods used to create and update entities, as well as methods to access entities. To generate session facade methods, select the checkbox for Generate Session Facade Methods in the Create Session Bean wizard, and use the following page to specify which methods to generate. JDeveloper automatically detects all the entities in the project and allows you to choose which entities and methods to use when creating session facade methods.

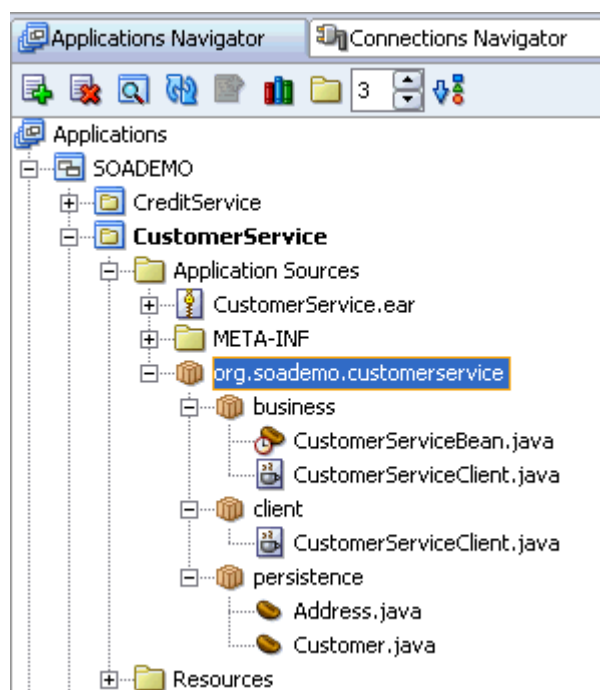
Figure 4–3 Selecting Session Facade Methods

You can generate session facade methods for every entity in the same project, which can be useful for testing purposes, but is often too much clutter in a single session bean. Session beans are often tailored to a specific task, and contain no more information than is required for that task. Use the tree control to explicitly choose which methods to generate.

Note: You can create session facade methods for any named queries on your entities. In [Figure 4–3](#), the "find" methods are named queries. When you generate session facade methods for a named query, you may have to specify the return type of the method. For lists and collections, this is not necessary, but for a single object you must specify the return type. For more information, see [Section 4.5.3.1, "How to Change the Return Type of a Method That Returns a Single Value"](#).

4.4.1.2 Choosing a Package

In a large application, having all of your objects in the same package can be very confusing. To organize your project you may want to partition your objects in separate packages. For example, in the Customer Service application in the SOA Order Booking application, there are separate packages for the business service (session bean), persistence (JPA entities), and client (Java test client). When you create a session bean, entity, or sample Java client, you have the option of changing the package name in the wizard or dialog, as shown in [Figure 4–4](#):

Figure 4–4 Location of packages in the Applications Navigator

4.4.1.3 Remote, Local and Web Service Interfaces

The type of interface required depends on the client. If the client is running in the same virtual machine (VM), a local interface is usually the best choice. If the client runs on a separate VM, a remote interface is required. Most Web applications (JSF/JSP/Servlet) have the client and service running in the same VM, so a local interface is the best practice. The Customer Service sample application runs everything locally in the same VM. Java clients (ADF Swing) run in a separate VM and require a remote interface.

If you have a stateless session bean that you will use as a web service, select to generate a web service endpoint interface.

4.4.2 What Happens When You Create a Session Bean

The session bean class contains session-wide fields and service methods. When you create a session bean, JDeveloper generates the bean class and a separate file for the local and/or remote interfaces. The remote interface is the name of the session bean, for example, CustomerService.java, while the bean class is appended with Bean.java and the local interface is appended with Local.java.

Note: You should not need to modify the interface files directly, so they are not visible in the Application Navigator. To view the interface files, use the System Navigator or the Structure Pane.

Example 4-3 CustomerServiceBean.java Bean Class

```
package org.soademo.customerservice.business;

import javax.ejb.Stateless;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
import org.soademo.customerservice.persistence.Customer;

@Stateless(name = "CustomerService")
public class CustomerServiceBean implements CustomerService,
                                           CustomerServiceLocal {
    @PersistenceContext(unitName = "customerServiceUnit" )

    private EntityManager em;

    public CustomerServiceBean() {
    }

    public Object mergeEntity(Object entity) {
        return em.merge(entity);
    }

    public Object persistEntity(Object entity) {
        em.persist(entity);
        return entity;
    }

    public Object refreshEntity(Object entity) {
        em.refresh(entity);
        return entity;
    }

    public void removeEntity(Object entity) {
        em.remove(em.merge(entity));
    }

    /** <code>select object(cust) from Customer cust where cust.custid =
:custid</code> */
    public Customer queryCustomerFindCustomerById(String custid) {
        return
        (Customer)em.createNamedQuery("Customer.findCustomerById").setParameter("custid",
custid).getSingleResult();
    }

    public String getCustomerStatus(String CustomerID) {
        return findCustomerById(CustomerID).getStatus();
    }

    public String addNewCustomer(Customer customer) {
        em.persist(customer);
        return "New customer added sucessfully to customer database";
    }
}
```

```

    public Customer findCustomerByEmail(String email, String password) {
        return
        (Customer)em.createNamedQuery("Customer.findCustomerByEmail").setParameter("email"
        ,
        email).setParameter("password",
        password).getSingleResult();
    }
}

```

4.4.3 What You May Need to Know When Creating a Session Bean

Typically you create one session facade for every logical unit in your application. A logical unit might be defined by its scope within the application. For example, create one session facade for administrative client operations and another session facade for customer client operations. How you create and name your session facades can facilitate UI development, so tailoring your session facades toward a particular task and using names that describe the task is a good practice.

4.4.4 How to Update an Existing Session Bean With New Entities

New session beans can be created at any time using the wizard. However, you may have an existing session bean that already contains custom implementation code that you want to update with new persistent data objects or methods.

To update an existing session bean with new entities:

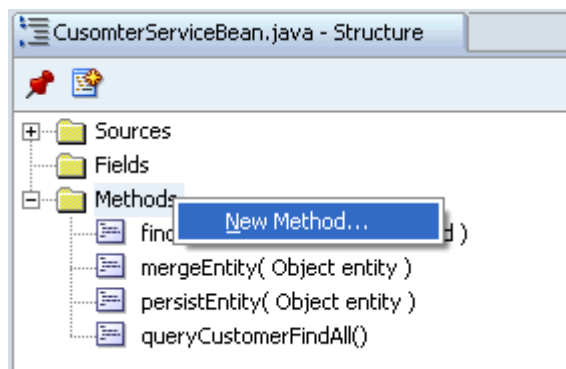
1. In the Navigator, right click the session bean and choose **Edit Session Facade**.
2. Use the Session Facade Options dialog to select the entities and methods to expose.

4.4.5 How to Add New Methods to the Session Bean

Once you've created your session bean and entities, you will probably want to create additional methods on the session bean.

The easiest way to create methods is to use the Bean Method Details dialog. Open this dialog by selecting the session bean in the Navigator, and then right clicking the Methods folder in the Structure pane, as shown in [Figure 4-5](#):

Figure 4-5 Using the context menu to add a method



Use the Bean Method Details dialog to specify various details, as shown [Figure 4-6](#):

Figure 4-6 Adding method details to the session bean

The dialog box, titled "Bean Method Details", contains the following fields and options:

- Name:** A text field containing "getCustomerStatus".
- Return Type:** A dropdown menu showing "java.lang.String" with a "Browse..." button to its right.
- Parameters:** A text field containing "String CustomerID".
- Throws:** An empty text field.
- Method Spec:** A text field containing "String getCustomerStatus(String CustomerID)".
- Expose options:** Three checkboxes:
 - ☒ Expose through Local interface
 - ☐ Expose through Remote interface
 - ☐ Expose through Web Service Endpoint interface
- Property tabs:** Three tabs labeled "Local Properties", "Remote Properties", and "Web Service Properties".
- Container Transaction Attribute:** A dropdown menu.
- Method Permissions:** A text field.
- Buttons:** "Help", "OK", and "Cancel" at the bottom.

Once you've created the method, you need to go to the source file (the bean class) and add implementation code. For example, in the SOA Order Booking application, the following code was used to get a customer's status:

Example 4-4 *getCustomerStatus* method

```
public String getCustomerStatus(String CustomerID) {
    return findCustomerById(CustomerID).getStatus();
}
```

4.5 Accessing Entities from a Session Bean

The entity manager is the client's gateway to entity management services offered by the EJB 3.0 persistence framework. The entity manager provides support for querying, updating, refreshing, and removing existing entity instances, and registering entity classes to create new persistent objects with identity.

The entity manager maintains a cache of instances within a transactional context called a *persistence context*. The persistence context allows the entity manager to track modified, created, and removed entity instances, and to reconcile entity instances with changes that were committed by external transactions concurrent with the entity manager's own transaction.

Entity instances queried through the entity manager may be freely passed to clients both inside and outside the EJB container. Clients may access and update the entity data as they would an ordinary Java object.

An `EntityManager` instance can be acquired both from within the EJB container (Java EE) and outside it (Java SE). This offers clients the flexibility to interact with persistent entity beans in a uniform way, without regard to whether the persistence code is running inside or outside the EE container.

4.5.1 How to Use the Entity Manager to Access Entity Data

Client sessions must obtain an `EntityManager` instance before interacting with persistent entity instances. In most cases, it's best to inject an instance of an `EntityManager`. However, you may need to use an `EntityManagerFactory` for more precise control, or use a JNDI lookup.

4.5.1.1 Injecting an EntityManager Instance

In [Example 4-5](#), container injection is used to obtain an `EntityManager` instance bound to the persistence unit that includes the `Customer` entity. The `EntityManager` is then used to persist a new `Customer` instance.

Note: [Example 4-5](#) assumes that an ID generator or other service exists to populate the primary key of the new instance.

Example 4-5 Session Bean Injected with an EntityManager Instance

```
@Stateless
public class CustomerManager {
    @PersistenceContext("customerServiceUnit")
    private EntityManager em;
    public void createCustomer() {
        final Customer cust = new Customer();
        cust.setName("First Last");
        em.persist(cust);
    }
}
```

4.5.1.2 Java SE Service Client Using an EntityManagerFactory

When container injection is not available, or when more control over the lifecycle of the `EntityManager` is desired, the client can obtain an `EntityManager` by first acquiring an `EntityManagerFactory`. The `javax.persistence.Persistence` class serves as a factory for acquiring an `EntityManagerFactory`. [Example 4-6](#) shows how a Java SE service client would obtain an `EntityManager`. In this example, an `EntityManagerFactory` is created and bound to the persistence unit that includes the `Customer` entity. Then an `EntityManager` instance is created from that factory and used to persist a new `Customer` instance.

Example 4-6 Java Service Facade Using EntityManagerFactory

```
public class CustomerService {
    public static void main(String[] args) {
        final EntityManagerFactory emf =
            Persistence.createEntityManagerFactory("customerServiceUnit");
        final EntityManager em = emf.createEntityManager();
        final Customer cust = new Customer();
        cust.setName("New Customer");
        em.persist(cust);
    }
}
```

4.5.1.3 Looking up the EntityManager Through JNDI

Another option available is to look up the EntityManagerFactory, or the EntityManager itself, through JNDI. [Example 4-7](#) shows how this is done from within a session bean.

Example 4-7 Looking up the EntityManager Through JNDI

```
@Stateless
@PersistenceContext(name="customerServiceUnit")
public class CustomerServiceBean implements CustomerService {
    @Resource SessionContext ctx;
    public void performService() {
        EntityManager em = (EntityManager)ctx.lookup("customerServiceUnit");
        ...
    }
}
```

4.5.2 How to Create a Named Query

To create a named query, you enter the query directly in the entity class using EJB QL syntax. Use `@NamedQueries` to denote a group of EJB QL queries, and `@NamedQuery` for each EJB QL query.

In [Example 4-8](#), three named queries are defined: `findAllCustomer`, `findCustomerById`, and `findCustomerByEmail`. The first find-all method is generated by default when you create the entity. The other two named queries were created by annotating the source code.

Example 4-8 Named Queries

```
@NamedQueries({
    @NamedQuery(name="Customer.findAll", query="select object(o) from Customer o"),
    @NamedQuery(name="Customer.findById", query="select object(cust) from Customer cust
    where cust.custid = :custid"),
    @NamedQuery(name="Customer.findByEmail", query="select object(cust) from Customer
    cust where cust.email = :email and cust.password = :password")
})
```

4.5.3 How to Expose Named Queries On the Session Facade

If you create or edit a session bean, you can choose which methods to expose through the session facade. See [Section 4.4.4, "How to Update an Existing Session Bean With New Entities"](#).

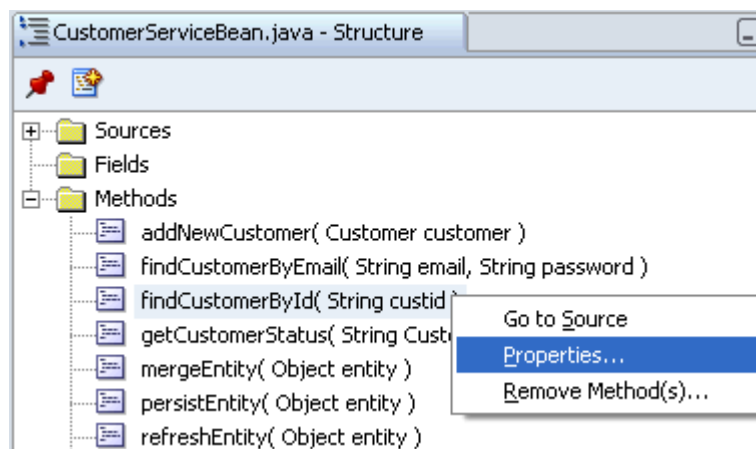
If the object returned is a list or collection, the framework determines the correct return type, but if it is a single object, you will need to specify the return type. See the following section, [Section 4.5.3.1, "How to Change the Return Type of a Method That Returns a Single Value"](#).

4.5.3.1 How to Change the Return Type of a Method That Returns a Single Value

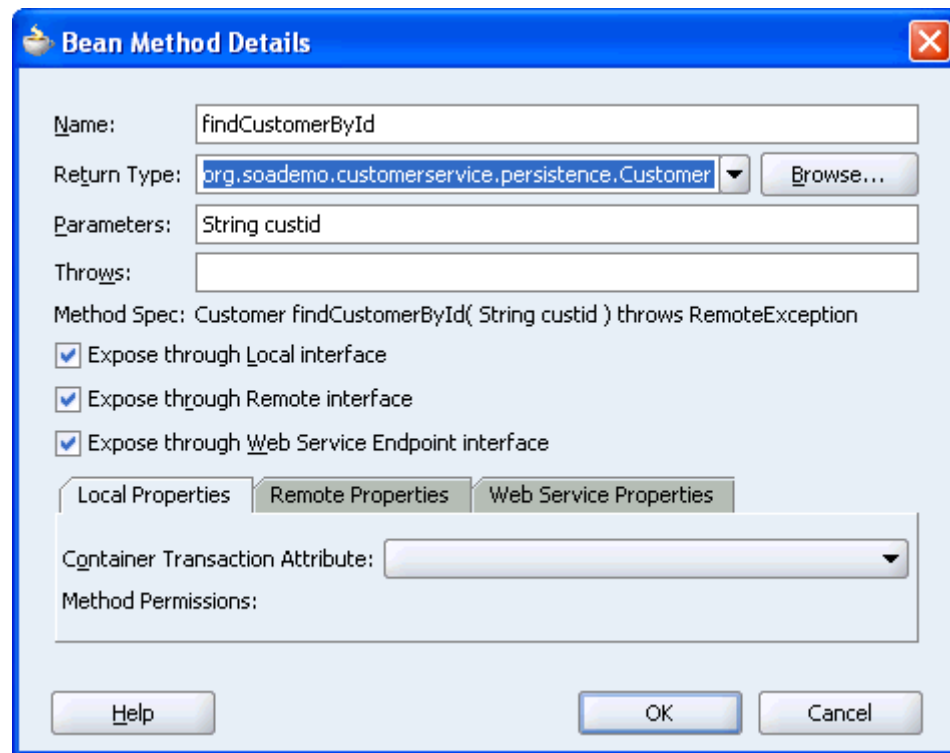
Changing the return type of a method is a two-step process: first change the return type in the Bean Method Details dialog, and then change the source code.

To open the Bean Method Details dialog, in the Structure pane, select the method you generated on the session facade, right-click it and choose **Properties**, as shown in Figure 4-7:

Figure 4-7 Changing properties of a bean method



In the Bean Methods Details dialog, change the return type so that it returns the proper type of object. For example, in the SOA Order Booking application, the `findCustomerId` method returns an instance of a `Customer`, located in the `org.soademo.customerservice.persistence` package, as shown in Figure 4-8:

Figure 4–8 Changing the return type on a session bean

Tip: Entering the return type by hand can introduce errors. If you are using one of the native Java types, use the drop-down list to select the proper return type. If the return type is a class, use the Browse button on the Bean Method Details dialog to locate the proper object in your workspace.

Once you've changed the return type of the method, you will also need to modify the source code. First, you must cast the object to the correct type, and then change `getResultList()` to `getSingleResult()`, as in [Example 4–9](#).

Example 4–9 Casting to a Customer type and returning a single result.

```
public Customer findCustomerById(Object custid) {
    return
    (Customer)em.createNamedQuery("findCustomerById").setParameter("custid",
    custid).getSingleResult();
}
```

4.6 Exposing a Session Bean as a Web Service

Exposing a session bean as a web service is fairly straightforward and most of the steps are automated when you use JDeveloper's wizards and dialogs. When you generate a web service endpoint interface for a session bean ([Section 4.4.1, "How to Create a Session Bean"](#)) the JSR-181 web services library is automatically added to your project and the `@WebService` annotation is added to your session bean. When you create methods using the Bean Methods Details dialog ([Section 4.4.5, "How to Add New Methods to the Session Bean"](#)), the appropriate annotations are added if you select **Expose through Web Service Endpoint interface**.

However, if you have an existing session bean or want to code the methods manually, you may do so. Following is the `CustomerService.java` file that shows the web services annotations.

Example 4–10 *CustomerService.java with web service annotations*

```
package org.soademo.customerservice.business;

import java.rmi.RemoteException;

import javax.ejb.Remote;

import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebService;

import org.soademo.customerservice.persistence.Customer;

@WebService(serviceName = "CustomerSvc",
            targetNamespace = "http://www.globalcompany.com/ns/customer")
public interface CustomerService {

    @WebMethod
    Customer findCustomerById(
        @WebParam(name = "custid",
            targetNamespace = "http://www.globalcompany.com/ns/customer")
        String custid) throws RemoteException;

    @WebMethod
    String getCustomerStatus(
        @WebParam(name = "CustomerID",
            targetNamespace = "http://www.globalcompany.com/ns/customer")
        String CustomerID);

    @WebMethod
    String addNewCustomer(
        @WebParam(name = "customer",
            targetNamespace = "http://www.globalcompany.com/ns/customer")
        Customer customer);

    @WebMethod
    Customer findCustomerByEmail(
        @WebParam(name = "email",
            targetNamespace = "http://www.globalcompany.com/ns/customer")
        String email,
        @WebParam(name = "password",
            targetNamespace = "http://www.globalcompany.com/ns/customer")
        String password);
}
```

4.7 Exposing Services with ADF Data Controls

The easiest way to bind services to a user interface is by using the ADF Data Control.

This section includes information on the following:

- [How to Create ADF Data Controls](#)
- [Understanding the Data Control Files](#)
- [Understanding the Data Control Palette](#)

Note: A SOA application does not require using the ADF Data Control. However, the ADF Data Control is useful for showing the products on the client and for testing purposes.

4.7.1 How to Create ADF Data Controls

To create an ADF data control from an EJB session bean, right-click a session bean in the Navigator and choose Create Data Control or drag a session bean onto the Data Control Palette.

Note: J2EE developers who do not want to rely on Oracle-specific libraries may use JSF managed beans instead of the ADF data control. This is more complex and beyond the scope of this book.

When you create a data control from an EJB 3.0 session bean, several XML files are generated and displayed in the Navigator. The generated files and the Data Control Palette are covered in the following sections.

4.7.2 Understanding the Data Control Files

When you create a data control, the following XML files are generated in the service layer:

- DataControls.dcx - data control definition file
- <session_bean>.xml - structure definition file
- ReadOnlyCollection.xml - design-time XML file
- ReadOnlySingleValue.xml - design-time XML file
- UpdateableCollection.xml - design-time XML file
- UpdateableSingleValue.xml - design-time XML file
- <entity_name>.xml - entity definition file, one per entity

How these files are related and used are covered in greater detail in Appendix A of the Oracle Application Development Framework Developer's Guide.

4.7.2.1 About the DataControls.dcx File

The DataControls.dcx file is created when you register data controls on the business services. The.dcx file identifies the Oracle ADF service layer adapter classes that facilitate the interaction between the client and the available business service. In the case of EJB, web services, and bean-based data controls, you can edit this file in the Property Inspector to add or remove parameters and to alter data control settings. For example, you can use the.dcx file to set global properties for various items, such as whether to turn on/off sorting.

4.7.2.2 About the Structure Definition Files

When you register a session bean as an Oracle ADF data control, an XML definition file is created in the service project for every session bean. This file is commonly referred to as the *structure definition* file. The structure definition file has the same name as the session bean, but has a.xml extension.

A structure definition is made up of three types of objects:

- Attributes
- Accessors
- Operations

4.7.2.3 About the Entity XML Files

When you create a data control, an XML file is generated for each entity (TopLink, EJB, or Java bean). These files are used for both ADF design-time and runtime. These files describe the structure of the class as well as UI hints, validators and labels for each attribute.

4.7.2.4 About the Design-time XML Files

Four files are generated solely for the design-time:

- ReadOnlyCollection.xml
- ReadOnlySingleValue.xml
- UpdateableCollection.xml
- UpdateableSingleValue.xml

These files are referenced by MethodAccessor definitions as the CollectionBeanClass which describes the available operations. Typically you do not edit this file by hand, but you could customize items on the Data Control Palette.

4.7.3 Understanding the Data Control Palette

Client developers use the Data Control Palette to create databound HTML elements (for JSP pages), databound Faces elements (for JSF JSP pages), and databound Swing UI components (for ADF Swing panels). The Data Control Palette comprises two selection lists:

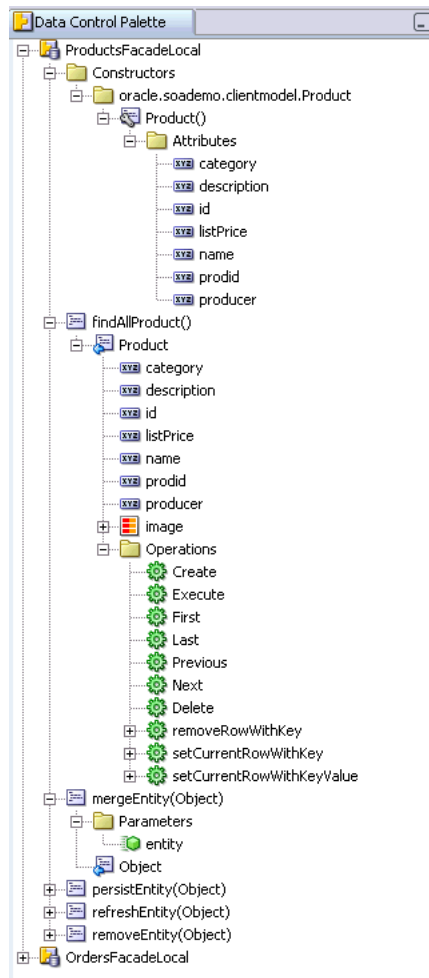
- Hierarchical display of available business objects, methods, and data control operations
- Dropdown list of appropriate visual elements that you can select for a given business object and drop into your open client document

Additionally, web application developers use the Data Control Palette to select methods provided by the business services that can be dropped onto the data pages and data actions of a page flow.

The Palette is a direct representation of the XML files examined in the previous sections, so by editing the files, you can change the elements contained in the Palette.

The hierarchical structure of the business services displayed in the Data Control Palette is determined by which business services you have registered with the data controls in your service project. The palette displays a separate root node for each business service that you register.

Figure 4–9 Data Control Palette



4.7.3.1 Overview of the Data Control Business Objects

The root node of the Data Control Palette represents the data control registered for the business service. Proceeding down the hierarchy from the root data control node, the palette represents bean-based business services as constructors, attributes, accessors or operations:

- Constructors - Createable types are contained within the Constructors node. These types call the default constructor for the object.
- Attributes - such as bean properties, which can define simple scalar value objects, structured objects (beans), or collections.
- Accessors - get() and set() methods.
- Operations - such as bean methods, which may or may not return a value or take method parameters. For Web Services, the Data Control Palette displays only operations.

For more information on using the Data Control Palette, see Oracle Application Development Framework Developer's Guide.

4.7.3.2 Refreshing ADF Data Controls After Modifying Business Services

After you have already created the data control definition for your service project, you may decide to update the data control after modifying your business services. Refreshing the data control definition makes the latest business service changes available to the ADF application.

The action you take to refresh the data control definition depends upon the type of change to the service project.

4.7.3.2.1 Viewing modified data controls in the Data Control Palette: If the palette is not yet displayed, select the View menu and choose Data Control Palette. If the palette is already displayed, right-click in the palette and choose Refresh.

4.7.3.2.2 Refreshing a data control definition for business services you have modified In the service project, define the new properties of the bean or other business service you want to create. Compile the .java file to regenerate the business service's metadata in its corresponding .xml file. If the modified business service is bean-based (such as an EJB session bean), you can refresh the data control by right clicking the session bean and choosing **Create Data Control**.

In the case of ADF Business Components, the data control definition is automatically updated whenever you make changes to your ADF BC project files.

4.7.3.2.3 Removing a data control definition for business services that have been removed: To remove a data control definition, in the view project, select the DataBindings.dcx file and in the Structure window, select the data control node that represents the business service that no longer appears in your service project. Right-click the data control node and choose Delete.

JDeveloper updates the data control definition file (DataBindings.dcx) in the service project. The DataBindings.dcx file identifies the Oracle ADF service layer adapter classes that facilitate the interaction between the client and the available business services.

4.7.3.2.4 Updating a data control after renaming or moving a business services In the service project, if you rename your business service or move it to a new package, you must update the reference to the service project in the client's data control definition.

In the view project, select the DataBindings.dcx file. In the Structure window, select the data control node that represents the moved business service. In the Property Inspector, edit the Package attribute to supply the new package name.

4.8 Testing EJBs

There are a few ways you can test your EJBs both inside and outside the container, depending on what you want to accomplish. The easiest way to test your EJBs is to create a Java service facade. A Java service facade can run outside the Java EE container, so you will not need to deploy the application or create a client. For testing inside the EE container, you can use JDeveloper's embedded server or you can deploy to an application server and test it remotely.

4.8.1 Testing Entities Outside the Container with a Java Service Facade

The standard method for testing your EJB services is to first deploy your application to an application sever, and then create a client to call methods through the session facade. However, you do not have to deploy an application to test your EJBs, you can do so within JDeveloper by creating a *Java service facade*. A Java service facade is a plain Java class with facade methods (CRUD methods and finder methods on entities accessed through the entity manager) and a runnable main() method. In order to test EJBs outside the container, you will also need to configure a persistence unit to run outside the container.

To test EJBs outside the container:

1. [Create a Persistence Unit that Runs Outside the Container.](#)
2. [Create a Java Service Facade.](#)
3. [Run the Java Service Facade.](#)

4.8.1.1 Create a Persistence Unit that Runs Outside the Container

The first step to testing your EJBs outside the container is to create a new persistence unit that is configured to run outside the container.

To create a a persistence unit that runs outside the container:

1. In the Navigator, right click the **persistence.xml** file and choose **New Persistence Unit**.
2. In the Create Persistence Unit dialog, name your persistence unit so that it will be easy to identify.

Tip: A persistence unit stores information about the service and connection, so naming your persistence unit something like "CusomterService-Outside" is often helpful.

3. Select the checkbox for **Outside Java EE Container**.
4. Click **OK** to save your changes.

[Example 4-11](#) shows the persistence.xml file from the Customer Service sample application, with a new persistence unit created to run outside the container:

Example 4-11 A persistence.xml file with a persistence unit that runs outside the container.

```
<?xml version="1.0" encoding="windows-1252" ?>
<persistence xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd"
             version="1.0" xmlns="http://java.sun.com/xml/ns/persistence">
  <persistence-unit name="CustomerService"/>
  <persistence-unit name="CustomerService-Outside">
    <class>org.soademo.customerservice.persistence.Address</class>
    <class>org.soademo.customerservice.persistence.Customer</class>
    <properties>
      <property name="toplink.jdbc.driver" value="oracle.jdbc.OracleDriver"/>
      <property name="toplink.jdbc.url"
                value="jdbc:oracle:thin:@Database_name:Port:SID_value"/>
      <property name="toplink.jdbc.user" value="SOADEMO"/>
      <property name="toplink.jdbc.password"
                value="A0D145248EE31051EAA1C822B80C610A"/>
      <property name="toplink.target-database" value="Oracle"/>
      <property name="toplink.logging.level" value="FINER"/>
    </properties>
  </persistence-unit>
</persistence>
```

4.8.1.2 Create a Java Service Facade

To create a Java service facade, you will use the Create Java Service Facade wizard.

To create a Java service facade:

1. In the Navigator, right click the **persistence.xml** file and choose **New Java Service Facade**.
2. On step 1 of the wizard, in the **Persistence Unit** list, make sure to select the persistence unit you created that runs outside the container, and select the checkbox for **Generate a main() method with sample code**.
3. On step 2, use the tree control to select the service facade methods you want to test and then click **Finish**.

Tip: Each time you create a Java service facade, JDeveloper will create a new service facade with the name `JavaServiceFacade`. If you want to save the service facade, rename it each time you generate it.

4.8.1.3 Run the Java Service Facade

The final step is to add code to the `main()` method and run the service facade.

To run the Java service facade:

1. In the Navigator, double click the Java service facade class to open it in the Java Source Editor.
2. In the Editor, scroll to the `main()` method and create a new method that calls any of the session facade methods you generated in [Section 4.8.1.2, "Create a Java Service Facade"](#).
3. Right-click the file in the Editor and choose **Run**.

For example, the following code is the `main()` method from a Java service facade that has been modified to print a list of all customer email addresses.

Example 4–12 Java service facade with method that prints email addresses

```
public static void main(String [] args) {
    final JavaServiceFacade javaServiceFacade = new JavaServiceFacade();
    // TODO: Call methods on javaServiceFacade here...
    List<Customer> customerList = javaServiceFacade.queryCustomerFindAllCustomer();
    for (Customer cust : customerList) {
        System.out.println(cust.getEmail());
    }
}
```

4.8.2 Testing Inside the EE Container Using the Embedded Server

JDeveloper has an embedded server that simplifies testing your EJBs. You do not have to deploy your application remotely to test your session beans.

To test EJBs using the embedded server:

1. [Create a Sample Java Client](#).
2. [Run the Service and Client](#).

4.8.2.1 Create a Sample Java Client

The easiest way to test your EJBs is to create a Java client and call methods on it. JDeveloper has a Sample Client utility that will generate a Java client and facade methods.

To create a sample Java client:

1. In the Navigator, right-click a session bean and choose **New Sample Java Client**.
2. In the Create Sample Java Client dialog, make sure **Connect to OC4J Embedded in JDeveloper** is selected and click **OK**.
3. In the Java Source Editor, uncomment the methods you want to test or create your own.

4.8.2.2 Run the Service and Client

You must first start the service by running the session bean, then you can run the client.

To run the service and client:

1. In the Navigator, right-click the session bean and choose **Run**. This may take a moment to initialize.
2. In the Navigator, right-click the sample Java client and choose **Run**.

You should see output in the Log window as your client runs.

4.9 Deploying EJB Applications

To deploy an EJB application, first you create a deployment profile, and then you deploy the project.

To create a deployment profile:

1. In the Navigator, right-click a project and choose **New**.
2. In Categories, expand General, select Deployment Profiles, and in the Items list, select EJB JAR File. Click OK.
3. In the Create Deployment Profile dialog, enter a name and location and click OK.
4. In the EJB JAR Deployment Profiles Properties dialog you may change properties or accept the defaults. Click OK.

To deploy the project:

1. In the Navigator, expand the folder for the project and the Resources folder within.
2. Right-click the .deploy file and choose how you want to deploy the application. If this is the first deployment, you will want to deploy to a **New Connection**.

JDeveloper compiles the application and creates JAR and EAR files in the <project>\deploy directory. If there are no errors, it displays the Configure Application dialog.

3. In the Configure Application dialog, click **OK**.

Deployment information will be output in the JDeveloper Log window.

Building Web Services

This chapter provides an overview of developing web services with Oracle JDeveloper 10g, as well as describing how to use web services to provide some of the functionality in the SOA Order Booking application.

This chapter includes the following sections:

- [Section 5.1, "Introduction to Web Services and SOA"](#)
- [Section 5.2, "Creating Web Services in JDeveloper"](#)
- [Section 5.3, "Using Web Services in JDeveloper"](#)
- [Section 5.4, "Debugging, Testing and Analyzing Web Services in JDeveloper"](#)

5.1 Introduction to Web Services and SOA

Web services is the term for a technology that consists of a set of messaging protocols and programming standards that expose business functions over the Internet using open XML-based standards. An individual web service is a discrete reusable software component that is accessed programmatically over the Internet, using HTTP/HTTPS but you can also invoke them using other protocols, such as JMS or SMTP, to return a response.

Web services allow enterprises to expose business functionality irrespective of the platform or language of the originating application, because the business functionality is exposed in such a way that it is abstracted to a message composed of standard XML constructs that can be recognized and used by other applications.

Web services are modular business services that can be easily integrated and reused, and it is this that makes them ideally suited as components within service-oriented architecture (SOA). OracleAS SOA helps you to create top-down and bottom-up web services or to create web services from existing functionality. You can do this using JDeveloper, alternatively you use Ant or integrate the Oracle SOA Suite with another tool. This chapter describes how to use JDeveloper as part of the SOA Suite to develop and use web services.

JDeveloper provides wizards and other tools to develop web services that are J2EE 1.4-compliant, with the ability to create JAX-RPC clients and services. In addition JDeveloper automatically generates web service metadata tag annotations into implementation classes (for compliance with JSR-181). There is support for defining:

- custom handlers
- serializers and deserializers
- MIME/DIME attachments
- WS-Security, security settings including SAML for developing SSO web service applications
- WS-Reliability to set quality of service
- WS-Management to set logging properties
- WSIF bindings that allow the publishing of Java, EJB, Java Message Service (JMS), and enterprise applications connected via Java Connector Architecture (JCA) adapters
- REST (Representational State Transfer)-based service

Web services can then be incorporated into SOA applications. The service is linked to a BPEL process (orchestration) (see [Chapter 7, "Using Oracle BPEL Process Manager"](#)), and ESB provides integration for the application (see [Chapter 6, "Using Oracle Enterprise Service Bus"](#)). Web services security is defined using OWSM (see [Chapter 10, "Securing the System"](#)).

Before describing how to use JDeveloper to create, test, and deploy web services, or alternatively, use existing web services in an application, it is useful to describe the XML standards on which web services are based.

5.1.1 Simple Object Access Protocol

The Simple Object Access Protocol (SOAP) is an XML-based protocol that is used for the sending and receiving of messages over a transport protocol, usually HTTP or SMTP. The SOAP specification, which you can read at the web site of the World Wide Web Consortium, at www.w3c.org/TR/SOAP, provides a standard way to encode requests and responses. It describes the structure and data types of message payloads using XML Schema.

A SOAP message is constructed of the following components:

- A SOAP envelope comprising:
 - The SOAP Header (optional) which contains behavioral information used for security, reliability, and so on.
 - The SOAP body, which contains the message exchanged between the client and server.
- A protocol binding that specifies how the SOAP envelope is sent. JDeveloper creates web services that use HTTP, and it also can generate WSIF bindings.

Web services are created using SOAP, the XML protocol for expressing data as XML and transporting it across the Internet using HTTP, and SOAP allows for more than one way of converting data to XML and back again. JDeveloper supports SOAP RPC encoding, SOAP RPC-literal style, and document-literal style (also known as *message style*).

JDeveloper creates standard J2EE 1.4 platform services that use JAX-RPC.

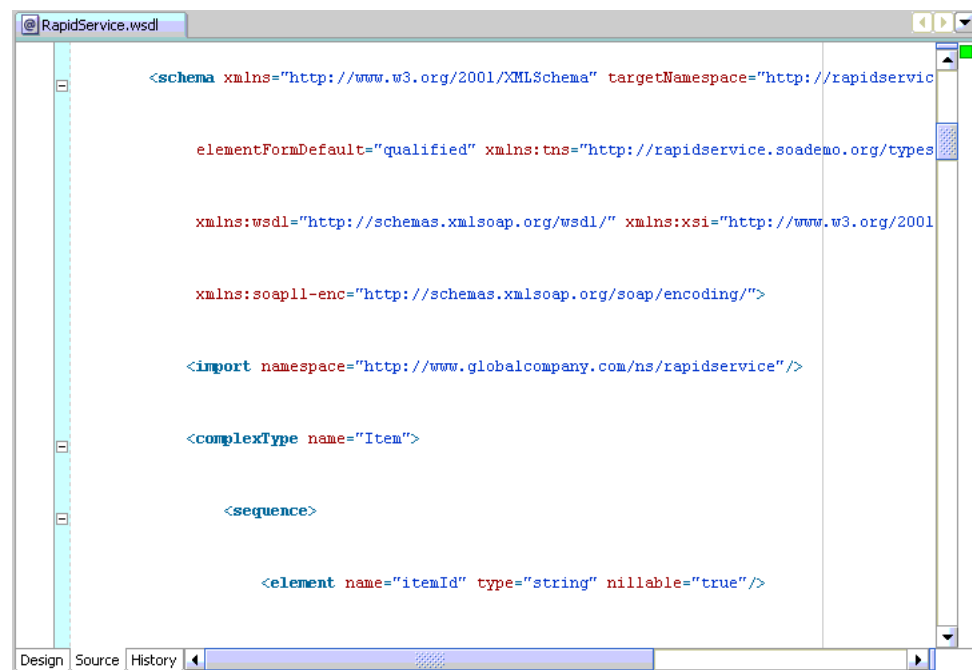
5.1.2 Web Services Description Language

The Web Services Description Language (WSDL) is an XML language used to describe the syntax of web services interfaces and their locations. Each web service has a WSDL document that contains all the information needed to use the service, the location of the service, its name, and information about the methods that the web service exposes. Alternatively, you can use a WSDL as the starting point for a top-down web service.

When you use one of JDeveloper's web service publishing wizards to produce your web service, the WSDL document for your service is automatically generated.

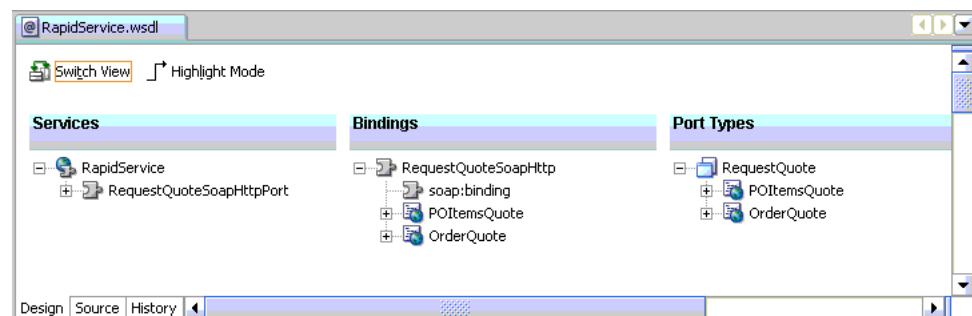
Figure 5–1 below shows part of the WSDL document generated by JDeveloper for the RapidService web service. It provides an illustration of the XML structure of a WSDL document.

Figure 5–1 *RapidService.wsdl in JDeveloper's XML Editor*



You can also view and edit a WSDL in design view as shown in Figure 5–2.

Figure 5–2 *RapidService.wsdl in JDeveloper's WSDL Editor*



The web service is described under the following headings: Services, Bindings, Port Types, Messages. You can reverse the order of these by clicking **Switch View**.

5.1.3 Universal Description, Discovery, and Integration

Universal Description, Discovery, and Integration (UDDI) provides a standards-based way of locating web services either by name or by industry category. UDDI registries can be public or private, such as a UDDI registry used within an organization.

Oracle Service Registry, which is part of Oracle SOA suite, provides a UDDI v3-compliant platform for publishing, categorizing and discovering web services and related resources across the enterprise. The Registry:

- Enables service providers to expose and advertise service offerings
- Allows service consumers to find, access and/or invoke services that meet defined criteria

Oracle Service Registry is integrated with the BPEL Process Manager, Oracle Web Services Manager and JDeveloper via the interoperable UDDI browsers in those products.

JDeveloper's UDDI browser, in the Connection Navigator, stores information about a UDDI registry and allows you to search a UDDI registry using search criteria that you specify to find web services that are described by WSDLs. JDeveloper comes with a preconfigured connection to the XMethods public UDDI registry, and you can also create your own registry connections to public or private UDDI registries. You can find the connection descriptor properties file, which contains the enquiry endpoint and the business keys of the registry, at `<jdev_install>/system<release_and_build_number>/uddiconnections.xml`, where `<jdev_install>` is the root directory in which JDeveloper is installed.

Note: If you have problems connecting to the preconfigured UDDI registry in JDeveloper or another external UDDI registry and you are behind a firewall, check the proxy server settings JDeveloper uses when you run web applications or access web services. In JDeveloper, choose **Tools > Preferences | Web Browser and Proxy**. Enter or change the proxy settings in this dialog. For more information, click **Help** the dialog.

JDeveloper's Find Web Service wizard browses UDDI registries to find web services by either name or category. You must have an appropriate connection from your machine so that JDeveloper can make a connection to the UDDI registry you select, for example, a connection the Internet if you want to search a public UDDI registry, and you can only generate a proxy to a web service that has a check in the "Is WSDL?" column, which identifies the registry entry as being defined by a WSDL document.

Figure 5–3 illustrates the report of a web service found in the XMethods UDDI registry using the Find Web Service wizard.

Figure 5–3 Report Produced by the Find Web Service Wizard

Web Service: Oracle Technology News Provider

Service Details

Service Name:	Oracle Technology News Provider
Service Key:	49E951DD-F1BE-C61C-9CCB-52AE3109FD9A
UDDI Registry:	http://uddi.xmethods.net/inquire

Description: Endpoint for service

Business Entity

Business Name:	mike.lehmann
Business Key:	6C8BFA5A-2BD6-3C62-1658-6C0294441F51

Service Bindings

Binding Details

Access Point:	http://otn.oracle.com/ws/oracle.otn.ws.news.OTNNews
Access Point Type:	HTTP URL

Design Source Preview

This figure illustrates the data structure types used in UDDI:

- Service details: This section gives information about the service, including the name.
- Business entity: This is the top-level data structure, called `businessEntity`, that contains information about the business providing the web service.
- Service bindings: This contains the `bindingTemplate` that contains information about the service access point, and the `tModel` that gives the technical specification of the web service.

5.2 Creating Web Services in JDeveloper

Creating and deploying web services, and finding them and incorporating them in your applications, is made simple in JDeveloper by the use of wizards.

Once you have created a web service, you can test it by running the service locally on JDeveloper's embedded OC4J server. Alternatively, you can deploy the service remotely to an instance of OC4J running outside of JDeveloper on your machine. There is more information about testing a web service in [Section 5.4, "Debugging, Testing and Analyzing Web Services in JDeveloper"](#).

5.2.1 How Java and Primitive Types Are Serialized to XML

Objects that can be passed to and from web services have to be capable of being serialized to an XML type, and then deserialized back to their original type. Oracle Web Services runtime is based on the JAX-RPC standard that defines how objects are serialized from Java2XML and XML2Java. JDeveloper automatically handles Java primitive types and certain Java standard types, converting them to XML types. Put simply, each parameter and return value of a web service method must conform to one of these rules:

- A Java primitive (for example `int`, `long`, `byte`), a Java primitive wrapper (for example `java.lang.Integer`), or `java.lang.String`.
- A Java bean with a zero-argument constructor, with a pair of "get" and "set" methods for each property to be exposed. Each property must itself conform to one of these rules.
- An array of a type that meets either of the first two rules.

For more information, see Appendix D, "Mapping Java Types to XML and WSDL Types" in *Oracle Application Server Advanced Web Services Developer's Guide*.

The wizards in JDeveloper allow you to extend the way Java objects are serialized by using custom mappings and serializers/deserializers. To find out more, search for "Setting Mappings between Java Methods and WSDL Operations" in the JDeveloper online help.

5.2.2 Web Service Development Tools in JDeveloper

Web services can be developed using two approaches:

- Bottom-up development
- Top-down development

There are two approaches for JAX-RPC bottom up web services development:

- Use one of JDeveloper's web services wizards, and point it at the implementation class. In the wizard, you select the methods you want to expose. JDeveloper generates all the necessary Java artifacts, deployment descriptors, and the WSDL without having to deal with manual operations.
- Use JSR-181 annotations in your Java classes or EJB session beans. The WSDL is generated, along with a number of helper files, for example, a mapping file. For more information, see the "Developing Contract-Driven Web Services Using JDeveloper" tutorial, which is available on the Oracle Technology Network at

<http://www.oracle.com/technology/obe/obe1013jdev/wstopdown/wstopdown.htm>

The bottom-up development approach is usually the easiest and quickest way of developing web services, but it may have some impact on interoperability. In the SOA Order Booking application, `RapidService` is developed bottom-up, and you can ensure interoperability by testing the WSDL for conformity to the WS-I Basic Profile 1.1. For more information, see [Section 5.4.2, "Analyzing Web Services for Interoperability"](#).

Top-down web services development starts from the WSDL. JDeveloper helps you to build WSDL to generate all the necessary Java artifacts and deployment descriptors, and the implementation Java file. Generally, the top-down approach is the best way of ensuring interoperability, since you start to develop your service describing the different message formats and operations using XML standards, so the resulting web service is platform agnostic. In the SOA Order Booking application, `CreditService` is developed top-down.

JDeveloper provides a number of wizards that help you develop and deploy web services. Each wizard creates the WSDL document and deployment files needed to publish your code as a web service.

- Create Java J2EE Web Service wizard, to expose methods in Java classes, or to expose methods in the remote interface of stateless session EJBs

Note: You can only use the Create Java J2EE Web Service wizard with EJB 2.0 and 2.1. For information about creating web services for EJB 3.0, see [Section 5.2.5, "How to Create a Web Service from EJB 3.0"](#).

- Create JMS J2EE Web Service wizard, to create web services that let you put messages on and take messages off a JMS destination
- Create PL/SQL J2EE Web Services wizard, to expose program units stored in PL/SQL packages
- Create J2EE Java Web Service from WSDL wizard, to create a top-down web service from an existing WSDL. Top-down web services development starts from the WSDL, and JDeveloper uses the WSDL to generate the necessary Java artifacts and deployment descriptors.

You can also create web services from ADF Business Components, either directly from the remote page of the application module editor, or by wrapping the business component as a stateless session EJB and using the Create Java J2EE Web Service wizard. For more information, search for "Publishing ADF Business Components Application Modules as Web Services" in the JDeveloper online help.

If you want to quickly try out some functionality you can create a web service which has default values from the context menu of a Java class.

5.2.3 How to Create a Bottom-Up Web Services

The Create Java J2EE Web Services wizard allows you to create web services starting from:

- Java classes
- Remote interface of EJB 2.0 or 2.1

Note: Web services for EJB 3.0 are created by annotating the Bean or interface. For more information, see [Section 5.2.5, "How to Create a Web Service from EJB 3.0"](#).

- ADF Business Components service session beans wrapped as EJBs

The Create JMS J2EE Web Service wizard allows you to publish a web service that exposes JMS destinations.

The Create PL/SQL J2EE Web Service wizard allows you to publish a web service based on PL/SQL program units stored in PL/SQL packages.

5.2.3.1 JAX-RPC and Oracle SOAP web services

JDeveloper allows you to create web services that conform to JAX-RPC, or that conform to the earlier Oracle SOAP stack. These are called:

- J2EE 1.4 web services which can be deployed to J2EE application servers
- J2EE 1.3 web services which can only be deployed to the SOAP containers in OC4J. These conform to the earlier Oracle SOAP stack, and you cannot use any JAX-RPC features with them.

5.2.3.2 Creating and deploying web services

Create the web service. If necessary, edit the deployment profile to change the J2EE context root, and deploy the web service.

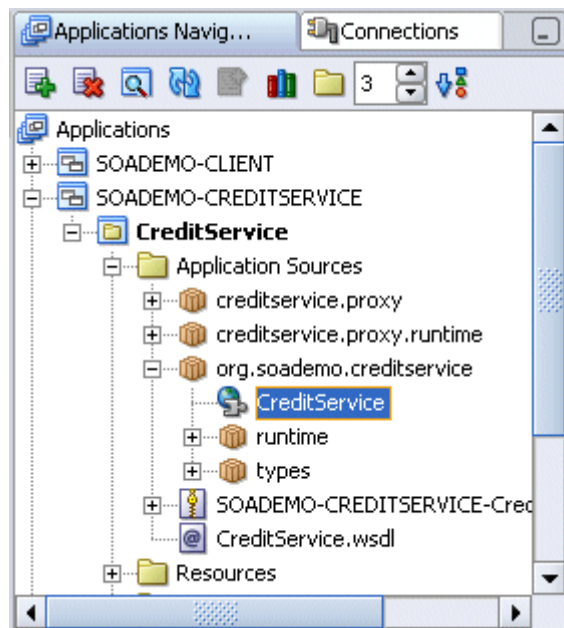
To create the web service:

1. In the Application Navigator, right-click the project and choose **New**.
2. In the New Gallery, in the **Categories** list expand Business Tier and select **Web Services**. From the **Items** list, select **Java Web Service**, **JMS Web Service** or **PL/SQL Web Service**, depending on the underlying implementation.
3. In the Select J2EE Web Service Version dialog, select the type of web service to create. This dialog only appears the first time you create a web service in a project, and the version you select is used for all web services you create in the project.
4. In the Create J2EE Web Service wizard, if the Welcome page is displayed, read the information, and click **Next**.
5. Complete the wizard.

For additional information about any of the pages in the wizard, click **Help**.

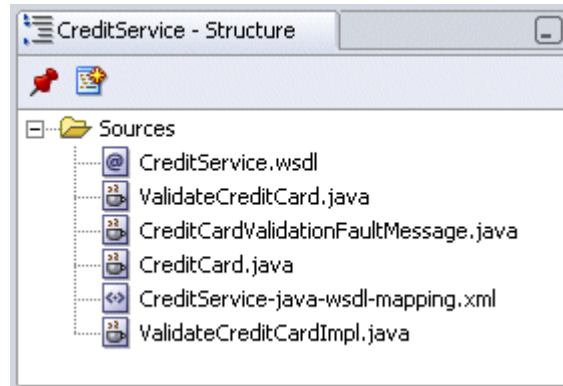
The web service is generated, and the generated artifacts are listed in the Application Navigator. This is illustrated in [Figure 5-4](#).

Figure 5-4 Web Service and Artifacts in the Application Navigator



CreditService is the generated web service container. When you select it the Structure window lists the files comprising the web service, see [Figure 5-5](#).

Figure 5-5 Web Service Files in the Structure Window



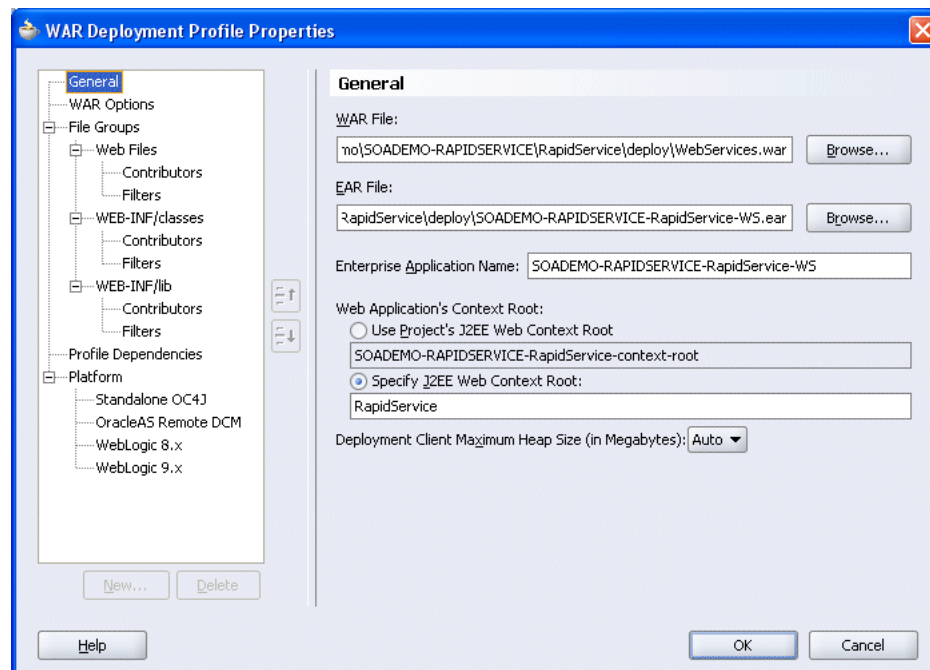
It is good practice to go examine the contents of the generated archive before you deploy the WAR. If you deploy using the default you may find that you are deploying redundant classes or other artifacts.

Another reason to examine the generated archive is to change the web context root. By default, the generated WAR deployment profile properties override the settings configured on the J2EE project settings panel in JDeveloper. Thus, the J2EE context root is inherited from the project. If you are following the SOA Order Booking application, you need to change the J2EE web context root before deploying the Rapid Manufacturer Service.

To examine and edit the WAR deployment profile:

1. In the Application Navigator, expand the project and the **Resources** folder.
2. Double-click `WebServices.deploy` to open the WAR Deployment Profile Properties dialog.
3. On the General page you can examine the files to be deployed, and if necessary add or remove some. This is also where you can change the J2EE Web Context Root.

The WAR deployment properties dialog is shown in [Figure 5-6](#).

Figure 5–6 WAR Deployment Profile Properties**To deploy the web service:**

1. In the Application Navigator, expand the project and the **Resources** folder.
2. Right-click `WebServices.deploy` to choose the application server connection.

5.2.4 Web Services metadata, JSR-181

One of the guiding principles for the next iteration of the Java Enterprise Platform (Java EE 5) is to provide ease-of-use features for the enterprise developer. To that end, the Web Services Metadata for the Java Platform Specification (JCP JSR 181) simplifies the Java web service programming model by providing a set of annotations that can be used to declaratively specify the web services of an application. The metadata annotates the Java source files that implement the web services, eliminating the need to modify and maintain additional service artifacts such as deployment descriptors and JAX-RPC mapping files. Thus, the process of developing and deploying a web service is greatly simplified.

When you create a web service from a Java class in JDeveloper, the runtime artifacts are generated into the `Web Content` folder.

However, if you use JSR-181 the annotations are processed at deployment time, and so the `Web Content` folder and its contents are not generated. For more information about JSR-181 see <http://www.jcp.org/en/jsr/detail?id=181>.

In the SOA Order Booking application, `RapidService` uses JSR-181. The JSR-181 Web Services library is added to the project and annotations are added at class and method level to the implementation class.

To add the JSR-181 library:

1. In the Application Navigator, right-click the project and choose **Project Properties**.
2. In the Project Properties dialog, choose **Libraries** and click **Add Library**.
3. In the Add Library dialog, select **JSR-181 Web Services** and click **OK**.
4. In the Project Properties dialog, ensure that **Export** is checked for **JSR-181 Web Services**, and click **OK**.

To add annotations to the Java class:

1. In the RequestQuote class, add the following as a class level annotation:

```
@WebService(serviceName="RapidService",targetNamespace="http://www.globalcompany.com/ns/rapidservice")
```

2. In the RequestQuote class, add the following as a method level annotation:

```
@WebMethod(operationName="OrderQuote")
```

5.2.5 How to Create a Web Service from EJB 3.0

If you want to create a web service from EJB 2.1, use the Create J2EE 1.4 Java Web Service wizard, which is invoked from the New Gallery. However, to create a web service from EJB 3.0, all you have to do is add the JSR-181 web service library to the project, add an annotation to the implementation class, and then deploy the EJB. The annotation is understood by the application server at runtime, and all necessary artifacts are generated server-side. The following example demonstrates this using SOADEMO-CUSTOMERSERVICE.

To create an EJB 3.0 web service:

1. Add the JSR-181 library. For more information, see [Section 5.2.4, "Web Services metadata, JSR-181"](#).
2. Locate the implementation class, in this case `CustomerService.java`, and open it in the source editor. Enter a class level annotation, for example:

```
@WebService(serviceName="CustomerSvc",
targetNamespace="http://www.globalcompany.com/ns/customer")
```

This is illustrated in [Figure 5-7](#).

3. Deploy the EJB to the application server. The annotation ensures that the server recognizes that it is a web service, creates a Web context based on the application name, and generates any necessary runtime artifacts.

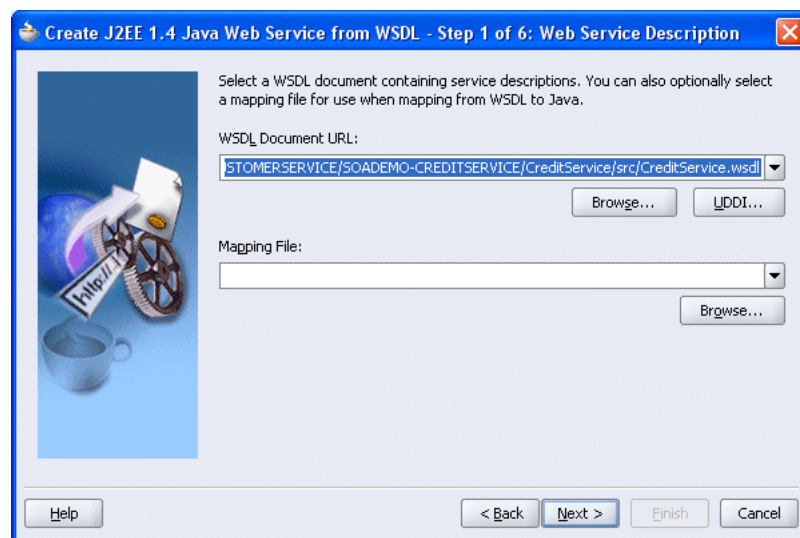
Figure 5–7 CustomerService.java

5.2.6 How to Create a Top-Down Web Service

A top-down web service is developed from a WSDL, where the WSDL provides the contract for the service. This is sometimes called the *contract driven* approach. For more information about developing a WSDL from the beginning, see the "Developing Contract-Driven Web Services Using JDeveloper" tutorial, which is available on the Oracle Technology Network at

<http://www.oracle.com/technology/obe/obe1013jdev/wstopdown/wstopdown.htm>

JDeveloper allows you to create a web service from a WSDL using the Create J2EE 1.4 Java Web Service wizard (shown in Figure 5–8), which is invoked from the New Gallery. In the SOA Order Booking application, CreditService is a backend service built from a WSDL.

Figure 5–8 Create J2EE 1.4 Java Web Service from WSDL

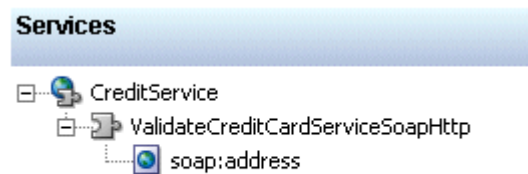
This wizard allows you to browse to a WSDL document on the local file system, enter the URL of a web service on the Internet or an intranet, or connect to a public or private UDDI registry to search for a web service. The web service is created using the contract information contained in the WSDL, and Java classes and other artifacts are automatically generated.

JDeveloper checks that the WSDL document is valid when you click **Next** from step 1.

You can examine an existing WSDL in JDeveloper by double-clicking it in the Structure window. By default, the editor opens in Design view showing the WSDL graphically. You can reverse the order of Services, Bindings, Port Types, Messages by clicking **Switch View**.

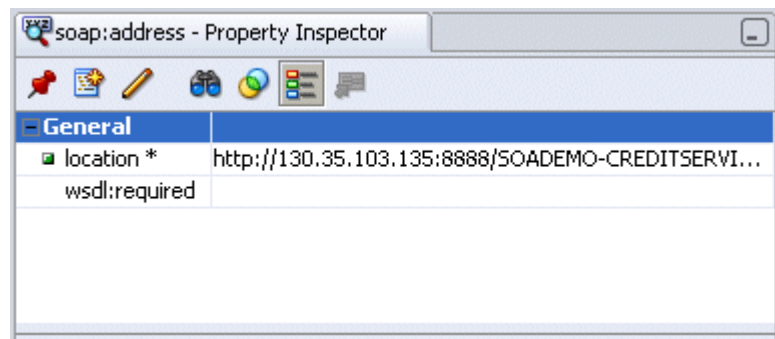
In Design view, expand the node under Services to show the `soap:address`, see [Figure 5-9](#).

Figure 5-9 *CreditService.wsdl in the Design Editor*



To see the actual value, select `soap:address` in the editor. The value is displayed in the property inspector, see [Figure 5-10](#).

Figure 5-10 *soap:address in the Property Inspector*



To see the WSDL as formatted XML in the editor, click the **Source** tab, see [Figure 5-11](#).

Figure 5–11 *CreditService.wsdl in Source Editor*

```

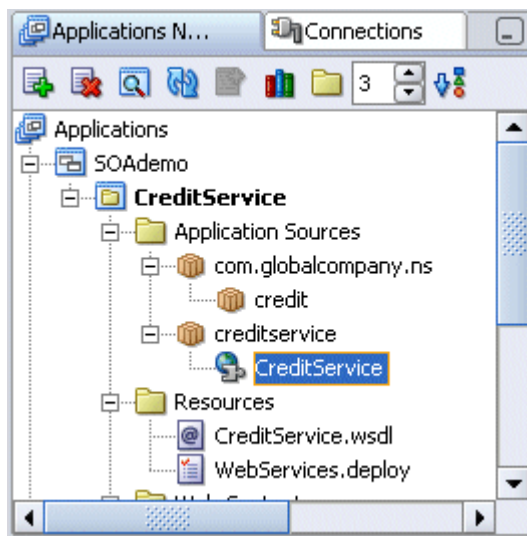
<service name="CreditService">
  <port name="ValidateCreditCardServiceSoapHttp"
        binding="tns:ValidateCreditCardSoapHttp">
    <soap:address location="http://localhost:8888/CreditService"/>
  </port>
</service>

```

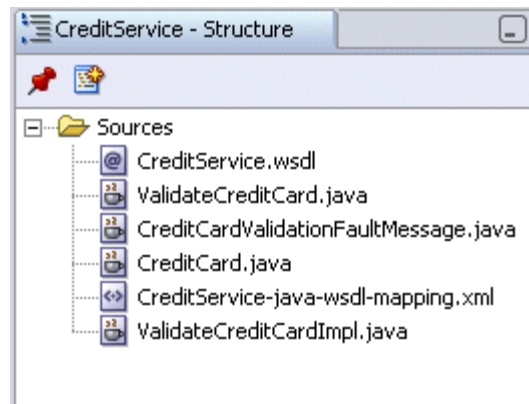
To create a top-down web service:

1. Open the New Gallery. Either:
Choose **File > New**, or
Click **Ctrl+N**
2. In the New Gallery dialog, expand **Business Tier**. Select Web Services and then double-click **Java Web Service from WSDL**.
3. In page Step 1 of 5: Web Service Description, enter or browse to the WSDL on which you want to base the web service.
4. Accept the defaults to complete the wizard.

The web service is generated (see [Figure 5–12](#)), and a deployment profile is created.

Figure 5–12 *Generated Web Service Container in the Applications Navigator*

CreditService is the generated web service container. When you select it the Structure window lists the files comprising the web service, see [Figure 5–13](#).

Figure 5–13 CreditService in the Structure Window**To deploy the web service:**

1. In the Application Navigator, expand the project and the `Resources` folder.
2. Right-click `WebServices.deploy` to choose the application server connection.

5.2.7 Deploying Web Services from JDeveloper

You can create web services for deployment either to Oracle Application Server, or to an external instance of JDeveloper's OC4J. The web service wizard generates a web archive (WAR) deployment profile names `WebServices.deploy` that you can deploy by just choosing **Deploy** from the context menu of the web service container in the navigator.

Note: Web services for EJB 3.0 are created by annotating the bean class. When the EJB is deployed, the web service artifacts are created server-side. For more information, see [Section 5.2.5, "How to Create a Web Service from EJB 3.0"](#).

5.3 Using Web Services in JDeveloper

JDeveloper makes it easy to use a web service in your application. You create a proxy to the service using the Create Web Service Proxy wizard, and call the methods in the client class in your application. You can launch the wizard wherever you have located or created a web service. Alternatively, you can launch the wizard directly and just enter the URL for the web service, or use the Find Web Service wizard to locate web services in a UDDI registry from a UDDI connection in the Connection Navigator. The UDDI connection allows you to search a UDDI registry using either the name or the category of the service. When you search by name, you can enter all or part of a name and you can use wildcards.

JDeveloper can interoperate with web services developed using other web service architectures. The Create Web Service Proxy wizard detects the type of web service, automatically generating the correct stub whether the service is an RPC-encoded, a document-encoded, or a document-literal type service.

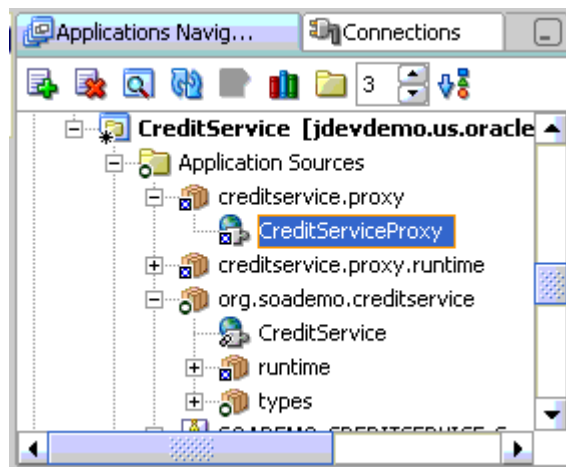
Generating a web service proxy in JDeveloper is a simple procedure. To invoke the Create Web Services Proxy wizard, in the New Gallery, expand **Business Tier** and choose **Web Services**, and in the **Items** list, choose **Web Services Proxy**.

- From a web service in JDeveloper. Choose **Generate Web Service Proxy** from the context menu of the web service container in the navigator. Alternatively, invoke the Create Web Services Proxy wizard and select or browse to the WSDL for the service.
- From a WSDL in the file system. In the Create Web Service Proxy wizard select or browse to the WSDL for the service.
- By pasting in the URL of the WSDL in the Create Web Service Proxy wizard when you have, for example, used Enterprise Manager to browse a running web service and discover the WSDL.
- From a public registry such as XMethods. When you find a web service in a public registry, copy the URL of the WSDL for the service. In step 1 of the wizard paste the URL in WSDL Document URL.
- From a public or private UDDI registry. In step 1 of the wizard, select UDDI to invoke the Find Web Services wizard.

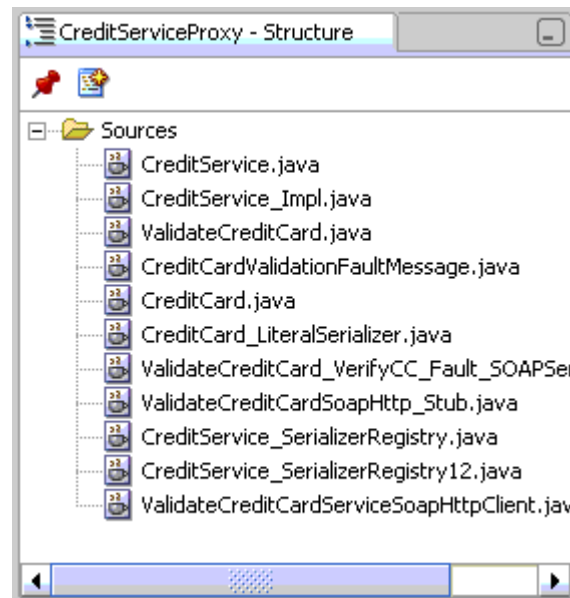
Figure 5–14 shows the Application Navigator and Structure window after generating a proxy to the `CreditService` web service.

Note: The SOA Order Booking application does not use a proxy to the `CreditService` web service.

Figure 5–14 *Generated Proxy in Application Navigator*



When you select the web service proxy container in the navigator, the files comprising the proxy are listed in the Structure window, see Figure 5–15. The file `<service_name>Client.java` is generated with a main method for testing the call to the service.

Figure 5–15 Files Comprising the Proxy in the Structure Window

Other types of web service proxies that JDeveloper can create are:

- **Sample Java client.** You can generate this client from the context menu of a web service container in the Application Navigator, or from the WSDL in the System Navigator. The sample Java client is designed for you to run against the web service running in the embedded OC4J server or against a remote instance of the OC4J server, and it contains a section for you to add your own code.
- **ADF data controls.** You can use web services in applications designed using Oracle ADF. When you create the data controls for a web service, behind the scenes JDeveloper creates a stub to the service that is configured to work with the rest of the ADF framework. You can create data controls for a web service by dragging the web service or WSDL from the navigator onto the Data Control Palette. Or you can use the context menu of the web service container or WSDL. Or you can create the data control from the New Gallery. Expand **Business Tier** and choose **Web Services**, and in the **Items** list, choose **Web Service Data Control**.

5.3.1 How to Create a Web Service Proxy

Before you build the page to collect the data, you must first generate a proxy based on the web service.

Note: You must have local access or know the URL for the WSDL in order to create the service proxy.

Tip: If you know the URL for the application server that hosts the WSDL, you can find the WSDL's URL using the WSIL Service Browser. The browser's URL is `http://<host_name>:<port>//inspection.wsil`.

To create a web service proxy:

1. In the Applications Navigator of JDeveloper, right click on a project, and choose **New**.

This opens the New Gallery.

2. From the Categories pane, expand the **Business Tier** node, and select **Web Services**.

Tip: If **Web Services** is not available under this node, use the **Filter By** dropdown menu to choose **All Technologies**.

3. From the Items pane, select **Web Services Proxy**.

This launches the Create Web Service Proxy Wizard.

4. Follow the instructions for the Wizard to create the proxy. Press F1 or **Help** for more information.

Tip: If you will be creating a data control for the web service, you must be sure to select the **Copy WSDL Into Project** option in the first step of the wizard. You will use this WSDL to create the data control.

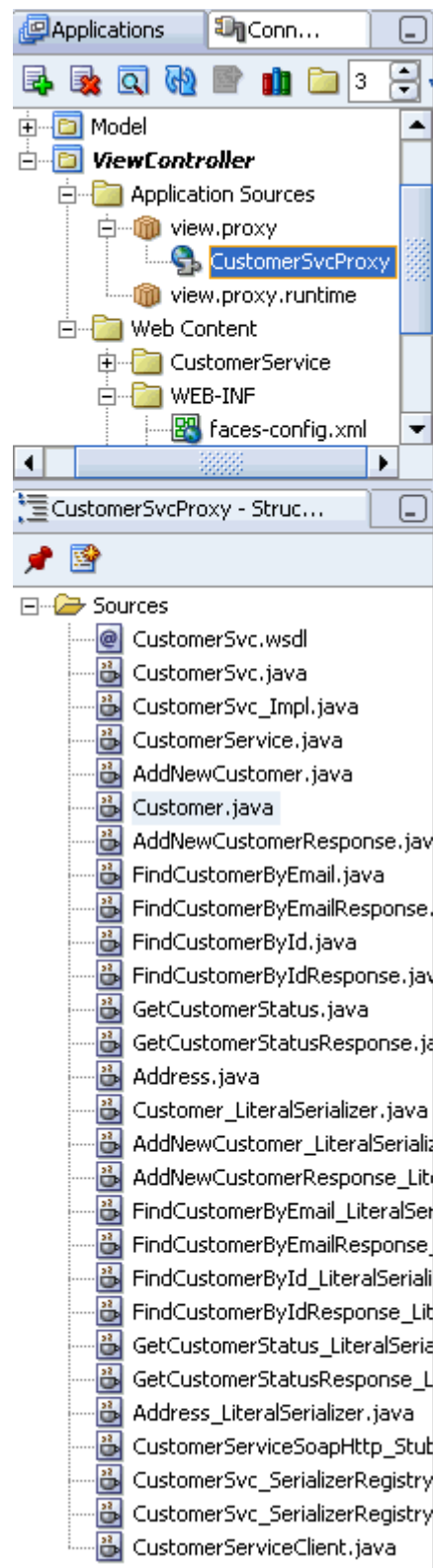
Once you complete the Wizard, you can use the WSDL to create a data control. You can then use the data control to declaratively create a JSF page that can collect the information. For more information, see [Section 5.3.3, "How to Create a Data Control From a Web Service"](#).

Note that the creation of data controls cannot take into account any complex data types. For example, in order to create a customer, an address might also be created. In this type of situation, you need to manually collect the data and invoke the service using a backing bean. For more information, see [Section 9.6.3, "How to Invoke a Service Using a Backing Bean"](#).

5.3.2 What Happens When You Create a Web Service Proxy

When you create a proxy to a web service using the Create Web Service Proxy Wizard, JDeveloper places the proxy by default into the `view.proxy` package in the Application Sources directory of the project. JDeveloper also generates source files based on the web service. You can view a list of these files in the Structure window when you click on the proxy in the Applications Navigator.

For example, [Figure 5–16](#) shows the list of source files created when you create a proxy based on the CustomerService WSDL.

Figure 5-16 *Generated Source Files for a Proxy are Displayed in the Structure Window*

You can view the generated code by double-clicking any of the classes in the window.

By default, JDeveloper opens the `<service-name>Client.java` file in the Source editor. This generated file provides access to each of the operations on the web service, and can be used to test the connection to the service, using its main method. For more information, see [Section 5.4, "Debugging, Testing and Analyzing Web Services in JDeveloper"](#). You can also use this class to invoke the web service directly. For more information, see [Section 9.6.3, "How to Invoke a Service Using a Backing Bean"](#). [Example 5-1](#) shows the generated code for the `CustomerServiceClient.java` file.

Example 5-1 CustomerServiceClient.java Code

```
public class CustomerServiceClient {
    private project_test.proxy.CustomerService _port;

    public CustomerServiceClient() throws Exception {
        ServiceFactory factory = ServiceFactory.newInstance();
        _port = ((project_test.proxy.CustomerSvc)factory.loadService(project_
            test.proxy.CustomerSvc.class)).getCustomerService();
    }

    /**
     * @param args
     */
    public static void main(String[] args) {
        try {
            project_test.proxy.CustomerServiceClient myPort = new project_
                test.proxy.CustomerServiceClient();
            System.out.println("calling " + myPort.getEndpoint());
            // Add your own code here

        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }

    /**
     * delegate all operations to the underlying implementation class.
     */

    public String addNewCustomer(Customer customer) throws
        java.rmi.RemoteException {
        return _port.addNewCustomer(customer);
    }

    public Customer findCustomerByEmail(String email, String password) throws
        java.rmi.RemoteException {
        return _port.findCustomerByEmail(email, password);
    }

    public Customer findCustomerById(String custid) throws
        java.rmi.RemoteException {
        return _port.findCustomerById(custid);
    }

    public String getCustomerStatus(String customerID) throws
        java.rmi.RemoteException {
        return _port.getCustomerStatus(customerID);
    }
}
```



```

/**
 * used to access the JAX-RPC level APIs
 * returns the interface of the port instance
 */
public project_test.proxy.CustomerService getPort() {
    return _port;
}

public String getEndpoint() {
    return (String) ((Stub) _port)._getProperty(Stub.ENDPOINT_ADDRESS
        _PROPERTY);
}

public void setEndpoint(String endpoint) {
    ((Stub) _port)._setProperty(Stub.ENDPOINT_ADDRESS_PROPERTY, endpoint);
}

public String getPassword() {
    return (String) ((Stub) _port)._getProperty(Stub.PASSWORD_PROPERTY);
}

public void setPassword(String password) {
    ((Stub) _port)._setProperty(Stub.PASSWORD_PROPERTY, password);
}

public String getUsername() {
    return (String) ((Stub) _port)._getProperty(Stub.USERNAME_PROPERTY);
}

public void setUsername(String username) {
    ((Stub) _port)._setProperty(Stub.USERNAME_PROPERTY, username);
}

public void setMaintainSession(boolean maintainSession) {
    ((Stub) _port)._setProperty(Stub.SESSION_MAINTAIN_PROPERTY,
        Boolean.valueOf(maintainSession));
}

public boolean getMaintainSession() {
    return ((Boolean) ((Stub) _port)._getProperty(Stub.SESSION_MAINTAIN_
        PROPERTY)).booleanValue();
}

/**
 * returns the transport context
 */
public ClientTransport getClientTransport() {
    return ((OracleStub) _port).getClientTransport();
}
}

```

If you chose to copy the WSDL into the project, the WSDL file is placed in a directory named for the service.

5.3.3 How to Create a Data Control From a Web Service

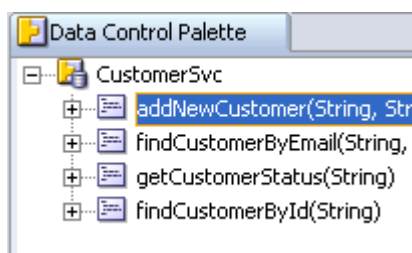
In order to create a data control for a web service, you must first have the WSDL file located in a project directory.

To create a data control from a WSDL file:

1. In the Applications Navigator, right-click the WSDL file (located in the directory named for the service, if you had the Create Web Service Proxy Wizard copy it for you).
2. From the context menu, choose **Create Data Control**.

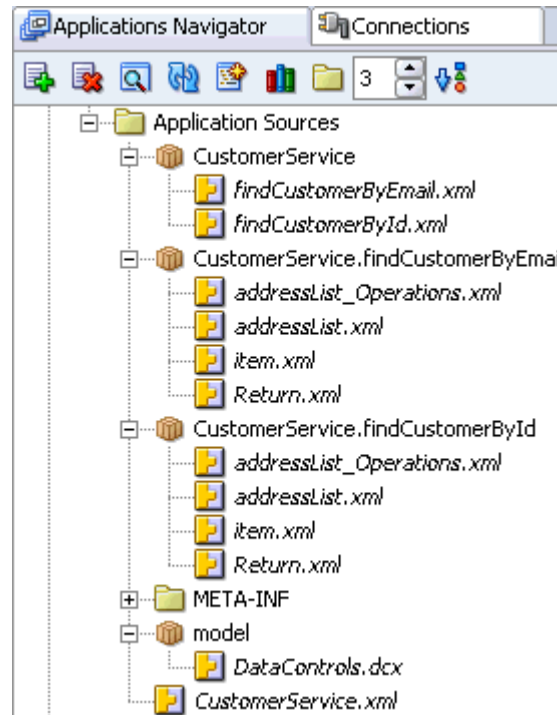
The data control is created and displays in the Data Control Palette. [Figure 5–17](#) shows the data control created from the `CustomerSvc` WSDL file.

Figure 5–17 Data Control Created From the CustomerSvc WSDL



5.3.4 What Happens When You Create and Use a Web Service Data Control

When you create a data control from a web service WSDL file, JDeveloper creates all the needed metadata files (such as the DCX file) needed to support the declarative creation of web pages. [Figure 5–18](#) shows the packages and XML files created for the CustomerService data control.

Figure 5–18 Generated Web Service Data Control Files

For more information about the different files that are generated when you create a data control, see [Section 4.7.2, "Understanding the Data Control Files"](#). For more information about how the data control is represented in the Data Control Palette, see [Section 4.7.3, "Understanding the Data Control Palette"](#).

5.4 Debugging, Testing and Analyzing Web Services in JDeveloper

You can test and analyze web services in a number of ways. In the life cycle of generating a web service, you could:

1. Test the deployed web service using Oracle Enterprise Manager. Every deployed service has its own test page, so you can quickly test that the service functions as you expect. For more information, see [Section 6.13.1, "How to Test ESB Services Using Oracle Enterprise Manager"](#)
2. Then test the service using a generated proxy. For more information, see [Section 5.3.1, "How to Create a Web Service Proxy"](#).
3. Perform more detailed testing using the HTTP Analyzer, which allows you to examine the messages sent and received by the service. For more information, search for "Examining Web Services using the Http Analyzer" in JDeveloper's online help.
4. Once you are satisfied that the service is working correctly, build the UI and retest, using the debugger for the application alongside the HTTP analyzer to examine the XML traffic.

If you are using an external web service in an application, you can check that it conforms to WS-I Basic Profile 2.0. For more information, see [Section 5.4.2, "Analyzing Web Services for Interoperability"](#)

5.4.1 Debugging Web Services in JDeveloper

JDeveloper has a fully featured debugger that allows you to debug web services locally on JDeveloper's embedded OC4J server or remotely, either on a deployed version of JDeveloper's OC4J server or on another remote server. For more information, search for "*About Debugging Web Services*" in JDeveloper's online help.

Note: Although you can debug a PL/SQL web service, what you are debugging is the Java class generated by JDeveloper to wrap the PL/SQL for deployment as a web service. Therefore, the correct way to ensure that a PL/SQL web service runs as expected is to debug the PL/SQL before you create a web service from it.

5.4.1.1 Locally Debugging a Web Service

Tip: You need to turn off JDeveloper's proxy settings before locally debugging the service. Remember to turn them back on when you have finished debugging.

When you locally debug a web service, you run the web service in JDeveloper's embedded OC4J server. There are two ways to locally debug the service, and in both cases you create a sample Java client for the web service. The sample Java client contains methods to run against each exposed method in the web service, and it acts as a proxy to the service. You can:

- Put breakpoints in the web service class, and then run a sample Java client against it. This procedure allows you to debug the service class itself.
- Put breakpoints in the sample Java client.

5.4.1.2 Remotely Debugging a Web Service

JDeveloper lets you debug a web service that is deployed remotely. The web service could be deployed to an external instance of OC4J on the local machine, or it could be a service located on a remote machine. In either case, you will need a connection to the remote server, and the remote server must be running in debug mode.

5.4.2 Analyzing Web Services for Interoperability

A key issue facing web services is how interoperable web services actually are. The important feature of web services is that they use common standards to avoid the problems that earlier solutions, for example CORBA, had. However, the standards themselves are being written at the same time as organizations are writing, deploying, and using web services. This has led to interoperability issues such as web services being written using different standards.

The Web Services-Interoperability Organization (WS-I) was formed by Oracle and other industry leaders to address these issues of interoperability, and to provide tools so that web services can be tested to see how well they interoperate. For more information about WS-I, including the specification, see the web site of The Web Services-Interoperability Organization (WS-I), at www.ws-i.org.

JDeveloper helps you to test the interoperability of web services by analyzing a web service for conformity to the WS-I Basic Profile 1.1. First you have to download a WS-I-compliant analyzer. This is available from the WS-I web site as well as from independent vendors. A set of test assertions is used to find out how well a web service conforms to the basic profile, and information is recorded about the following:

- Discovery when a web services has been found using a UDDI registry. If the service has not been found using the Find Web Services wizard, this section of the report returns errors in the Missing Input section.
- Description of a web service's WSDL document, where the different elements of the document are examined and con-conformities are reported. An example of a failure in this section is a failure of assertion WSI2703, which gives the message "WSDL definition does not conform to the schema located at <http://schemas.xmlsoap.org/wsd1/soap/2003-02-11.xsd> for some element using the WSDL-SOAP binding namespace, or does not conform to the schema located at [http://schemas.xmlsoap.wsd1/2003-02-11.xsd](http://schemas.xmlsoap.wsd1/wsd1/2003-02-11.xsd) for some element using the WSDL namespace."
- Message. This tests the request and response messages when the connection is made to the web services and it sends its reply.

To do this in JDeveloper, search for "WS-I Basic Profile" in JDeveloper's online help and follow the instructions.

Figure 5-19 shows part of the WS-I report of the Credit Service in the SOA Order Booking application.

Figure 5-19 Example of a WS-I Profile Conformance Report

WS-I Profile Conformance Report

Report: WS-I Basic Profile Conformance Report.
Timestamp: 2006-08-09T12:07:16.092

Copyright (c) 2002-2004 by [The Web Services-Interoperability Organization](http://www.ws-i.org) (WS-I) and Certain of its Members. All Rights Reserved.

Analyzer Tool Information

Version	1.0.0
Release Date	2005-07-04
Implementer Name	WS-I Organization
Location	http://www.ws-i.org

Analyzer Runtime Environment Information

Runtime Name	Java(TM) 2 Runtime Environment, Standard Edition
Runtime Version	1.5.0_08-b05
Operating System Name	Windows XP

Design | Source | Preview | History

The report produced is `wsi-report.html`, and it is located in the directory where the WS-I test tools are located. In this case, the report shows that the `CreditService.wsdl` is conformant and can be used for contract driven development of Credit Service in the SOA Order Booking application. See [Figure 5–20](#).

Figure 5–20 Summary Section of the WS-I report of `CreditService.wsdl`

Summary	
Result	passed
Artifact Targets Analyzed: The summary result applies to the following artifact targets which were specified in the analyzer configuration file.	
Description	port=ValidateCreditCardServiceSoapHttp
Message	null

5.4.3 Using the HTTP Analyzer

JDeveloper has a very useful tool, the HTTP Analyzer, that monitors and records the request/response TCP packets and allows you to see the contents of SOAP messages passed when a proxy connects to a web service and the service returns a response.

When you start the HTTP Analyzer, it updates the proxy settings in JDeveloper so that all TCP data is sent through an intermediate port and then on to your original proxy, if one is defined. The proxy settings are restored when you turn the HTTP Analyzer off, or when you exit JDeveloper.

Once you complete the Wizard, you can use the WSDL to create a data control. You can then use the data control to declaratively create a JSF page that can collect the information.

You need to have generated a proxy to a working web service. For more information, see [Section 5.3.1, "How to Create a Web Service Proxy"](#)

To use the HTTP Analyzer the web service:

1. Display the HTTP Analyzer. From the View menu, choose HTTP Analyzer.
2. Click the green button to start the analyzer.
3. Run the generated proxy. The HTTP Analyzer displays the request/response packets.

For further information, search for "*Using the Http Analyzer*" in JDeveloper's online help.

Using Oracle Enterprise Service Bus

This chapter describes how to create and configure services in Oracle Enterprise Service Bus to create a message instance flow. Testing the flow is also discussed in this chapter.

This chapter includes the following sections:

- [Section 6.1, "Introduction to Oracle Enterprise Service Bus"](#)
- [Section 6.2, "Creating an Oracle Enterprise Service Bus Flow"](#)
- [Section 6.3, "Creating Oracle Enterprise Service Bus Projects"](#)
- [Section 6.4, "Creating Oracle Enterprise Service Bus Systems or Service Groups"](#)
- [Section 6.5, "Browsing for Deployed Services Using the Service Explorer Browser"](#)
- [Section 6.6, "Adding Endpoint Properties for Adapter or SOAP Services"](#)
- [Section 6.7, "Adding SOAP Services to Oracle Enterprise Service Bus"](#)
- [Section 6.8, "Adding Adapter Services to Oracle Enterprise Service Bus"](#)
- [Section 6.9, "Adding Routing Services to Oracle Enterprise Service Bus"](#)
- [Section 6.10, "Specifying Routing Rules"](#)
- [Section 6.11, "Adding a Domain-Value Map to Oracle Enterprise Service Bus"](#)
- [Section 6.12, "Registering ESB Projects and Services with the ESB Server"](#)
- [Section 6.13, "Testing and Validating the Oracle Enterprise Service Bus Flow"](#)

For specific information on setting up Oracle Enterprise Service Bus components in the SOA demo, see *Oracle Application Server Tutorial*.

6.1 Introduction to Oracle Enterprise Service Bus

Oracle Enterprise Service Bus moves data among multiple endpoints, both within and outside of an enterprise. It uses open standards to connect, transform, and route business documents (as Extensible Markup Language (XML) messages), among disparate applications. It enables monitoring and management of business data, with minimal impact on existing applications. An enterprise service bus is the underlying infrastructure for delivering a service-oriented architecture (SOA) and event-driven architecture (EDA).

Oracle Enterprise Service Bus is the foundation for services using SOA and EDA. At its core, it is a loosely coupled application framework that provides your business with increased flexibility, reusability, and overall responsiveness in a distributed, heterogeneous, message-oriented environment using industry standards.

Oracle Enterprise Service Bus features that provide the ability to integrate applications fall into three broad categories:

- **Connectivity**

Connectivity is provided through adapter services and Simple Object Access Protocol (SOAP) invocation services. Oracle Enterprise Service Bus provides tools and wizards to assist you in invoking and creating services.

- **Document Transformation**

Oracle Enterprise Service Bus includes a standards-based data mapper that specifies an `xsl` file to transform data from one XML schema to another, thus enabling data interchange among applications using different schemas. Multiple transformations may be required to achieve the desired result. These transformations can be reused, as needed, across Oracle Enterprise Service Bus.

- **Routing**

Data contained within XML messages are distributed from the source application to a target application using routing services. As the name suggests, a *routing service* determines how a message gets from one point to another within the Oracle Enterprise Service Bus environment.

Minimally, *Routing rules* specify the set of services (referred to as *target services*) that Oracle Enterprise Service Bus will invoke when the routing service receives a message. In addition, when you configure routing rules, you can specify the following options:

- Whether a *filter expression*, which specifies that the contents (payload) of a message be analyzed before any service is invoked, be applied
- Whether a document transformation is applied
- Whether execution is synchronous or asynchronous
- The priority level for each routing rule specified in relation to the others
- The ESB systems, which are logical units of ESB services, from which messages are accepted.

Read this chapter to understand:

- How to create a message instance flow
- How to invoke external services
- How to create adapter services
- How to create routing services and configure routing rules
- How to test the Oracle Enterprise Service Bus flow

For an additional introduction to Oracle Enterprise Service Bus, see "Introduction to Oracle Enterprise Service Bus" in *Oracle Enterprise Service Bus Developer's Guide*

6.2 Creating an Oracle Enterprise Service Bus Flow

You can create an Oracle Enterprise Service Bus flow using the tools in Oracle JDeveloper ESB Designer. Oracle JDeveloper ESB Designer tools provide a visual approach that simplifies application development and greatly reduce coding tasks.

The basic steps to create the Oracle Enterprise Service Bus flow are:

1. Create the Oracle JDeveloper ESB Designer applications and ESB projects. See [Section 6.3, "Creating Oracle Enterprise Service Bus Projects"](#).
2. Create ESB systems and, optionally, service groups. See [Section 6.4, "Creating Oracle Enterprise Service Bus Systems or Service Groups"](#).
3. Create the inbound and outbound services, or invoke external services. See [Section 6.7, "Adding SOAP Services to Oracle Enterprise Service Bus"](#) and [Section 6.8, "Adding Adapter Services to Oracle Enterprise Service Bus"](#).
4. Create routing services and configure the routing rules. See [Section 6.9, "Adding Routing Services to Oracle Enterprise Service Bus"](#) and [Section 6.10, "Specifying Routing Rules"](#).
5. Register services in the ESB projects with the ESB Server. See [Section 6.12, "Registering ESB Projects and Services with the ESB Server"](#).

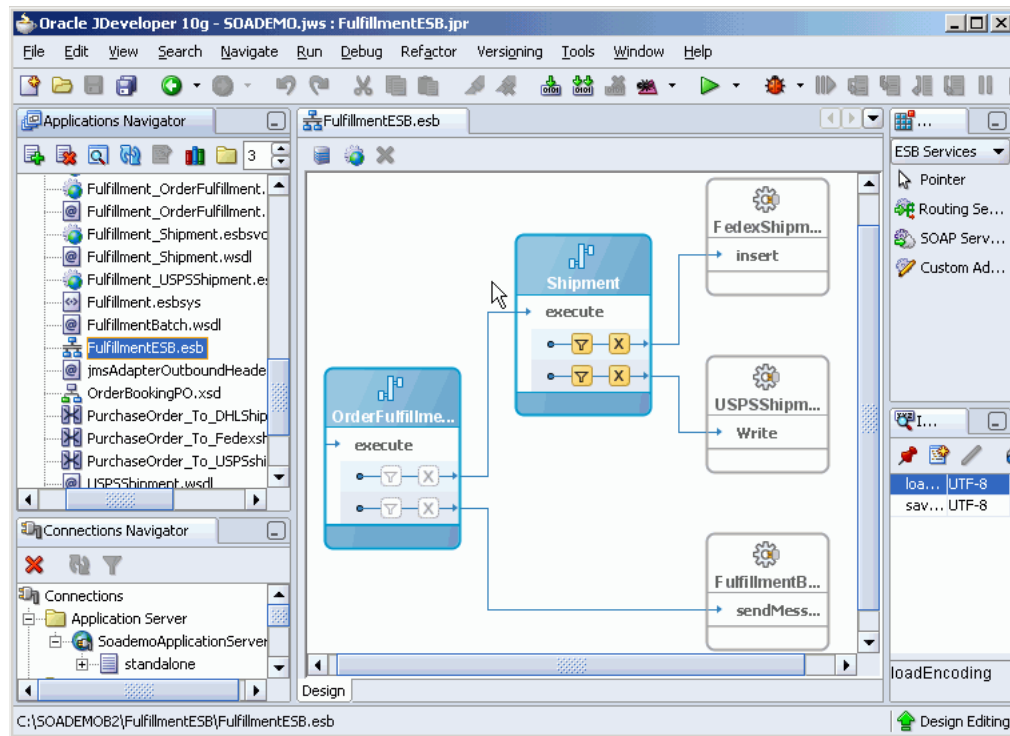
To begin creating an Oracle Enterprise Service Bus flow, you must create an application and then an ESB project. An application contains one or more projects. An ESB project is a collection of related files, for example, all the Oracle Enterprise Service Bus systems, services, and associated files within a single Oracle Enterprise Service Bus flow.

ESB systems and service groups are units for organizing the services for the Oracle Enterprise Service Bus flow.

You create an Oracle Enterprise Service Bus flow by dragging and dropping services from the Oracle JDeveloper Components Palette to the Design tab. These services include adapter, SOAP, and routing services.

The message instance flow can be specified as asynchronous or synchronous execution. Synchronous provides an immediate response to a request; asynchronous does not. For more information, see [Section 6.10.4, "How to Specify Synchronous or Asynchronous Execution"](#).

[Figure 6–1](#) is an example of a Design tab of an ESB system in Oracle JDeveloper ESB Designer, showing the service icons in the system.

Figure 6–1 Oracle JDeveloper ESB Designer

For additional information about developing an Oracle Enterprise Service Bus flow with Oracle JDeveloper ESB Designer, see "Developing the Enterprise Service Bus" in *Oracle Enterprise Service Bus Developer's Guide*.

6.2.1 What Happens When you Create an Oracle Enterprise Service Bus Flow

When the ESB project is registered with the ESB metadata server, ESB artifacts created in Oracle JDeveloper ESB Designer are registered in the design time metadata server.

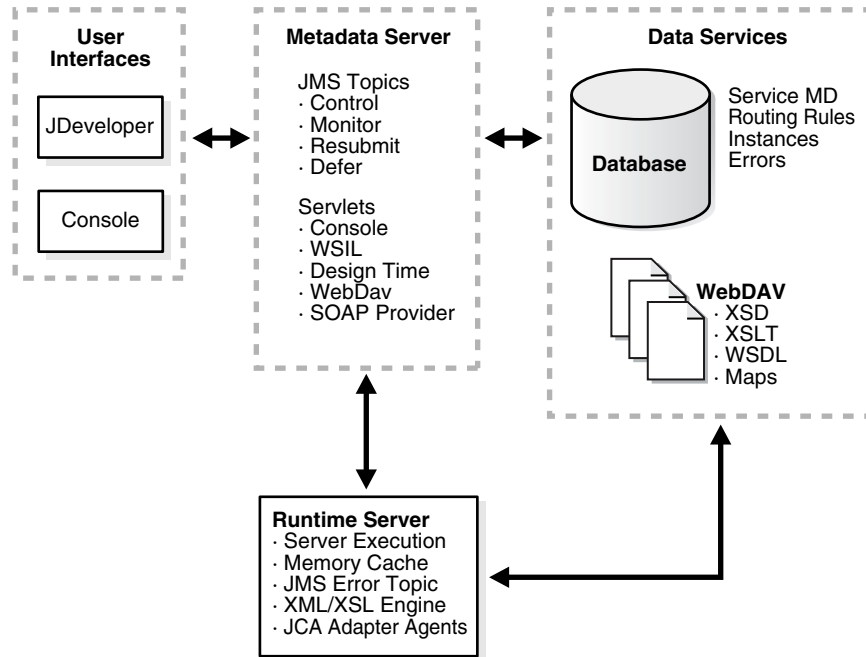
The ESB service definition files are translated and captured in a relational form in the ORAESB schema in the database repository while the XSD, XSL, and WSDL files are written to the file system. The service definitions in the database repository have pointers to the XSD, XSL, and WSDL files on the file system.

At startup, an ESB runtime server automatically accesses the design metadata server to cache the complete information for the ESB runtime services. During runtime the server automatically listens on the control topic in the metadata server for updates to the information, and refreshes the cache from the database repository if changes are found.

At runtime, the ESB message flow is initiated by an inbound adapter polling or listening for an event, such as a file copied to the directory specified for an inbound file adapter. The ESB flow is also initiated when an external SOAP/HTTP client/process invokes the Web service associated with an ESB service.

ESB enables you to modify some metadata at design time with Oracle ESB Control. These updates are automatically sent to the design time metadata server.

Figure 6–2 is an example of Oracle Enterprise Service Bus architecture in a single instance environment.

Figure 6–2 Oracle Enterprise Service Bus Architecture - Single Instance Environment

6.2.2 What You May Need to Know

When you create an ESB project and services in Oracle JDeveloper ESB Designer, files are generated for the ESB flow. [Table 6–1](#) lists the types of ESB files generated, a brief description of the file, and whether the file is used at design time or runtime.

Table 6–1 ESB Files

Type of File	Description	Design Time	Runtime
Project (esb)	Project file created whenever an ESB project is created in Oracle JDeveloper ESB Designer	yes	no
System (esbsys)	System file created whenever an ESB system is created in Oracle JDeveloper ESB Designer	yes	no
Service group (esbgrp)	Service group file created whenever an ESB system is created in Oracle JDeveloper ESB Designer	yes	no
Service (esbsvc)	Service definition file created whenever an adapter, routing, or SOAP service is created in Oracle JDeveloper ESB Designer	yes	no
Service (wsdl)	Service WSDL file	yes	yes
Adapter header (wsdl)	Inbound and outbound adapter header files created whenever an adapter service is created in Oracle JDeveloper ESB Designer	yes	yes
Toplink_mappings (xml)	Mapping file for DB adapter	yes	yes
Schema (xsd)	Schema table file created when a DB adapter is created in Oracle JDeveloper ESB Designer	yes	yes
Data mapping (xsl)	Transformation file created when a data transformation map is generated in Oracle JDeveloper ESB Designer	yes	yes

6.3 Creating Oracle Enterprise Service Bus Projects

To begin configuring an Oracle Enterprise Service Bus, you must create an ESB project in an application. An ESB project is a collection of related files, for example, all the ESB services and associated files within a single Oracle Enterprise Service Bus.

6.3.1 How to Create an ESB Project

After you have created an application, create the ESB project that will contain the ESB services using Oracle JDeveloper ESB Designer.

To create an ESB project:

1. In the Applications Navigator, right-click the application to which you want to add a project and then choose **New Project**.

The New Gallery opens.

2. Select **ESB Project** under **General Projects**, and then click **OK**.

The Create ESB Project dialog opens.

3. In the **Project Name** field, enter the name of the project.

4. In the **Directory Name** field, specify the directory where the project files will be stored.

Accept the default directory.

5. In the **Diagram Name** field, enter a name for the diagram that will schematically represent the project.

6. Click **OK**.

6.3.2 What Happens When You Create an ESB Project

When an ESB project is created, an entry appears in the Application navigator under the application when a project is created, in the form *project_name.esb*. You can double-click on the entry in the Application navigator or right-click on the entry and select Open from the menu to view the ESB project in the Design tab. The actual source of the project file is not visible in Oracle JDeveloper ESB Designer.

A design area is presented with a label of the name you specified for the ESB project. You use this design area to model the enterprise service bus system. It enables you to drag and drop ESB Services (adapter, routing service, SOAP service, and custom adapter) from the Component Palette (in the upper-right corner) into the design area.

6.4 Creating Oracle Enterprise Service Bus Systems or Service Groups

ESB systems and service groups are units for organizing the services for the Oracle Enterprise Service Bus.

6.4.1 How to Create an ESB System or Service Group

After you have created an ESB project, you can create an ESB system, or Service Group, using Oracle JDeveloper ESB Designer.

To create an ESB system or group:

1. In the Applications Navigator, open the ESB Project in which you want to create the ESB system or service group.
2. In the **Design** tab, click the **Create System/Group** icon on the top of the diagram window.

The Create ESB System or Service Group dialog opens.

3. In the Create ESB System, or Service Group, dialog, follow these steps:
 - a. For the **Create** option, select **System** or **Service Group**.
 - b. In the **Name** field, enter a unique name. The name must be unique across the ESB project in which you are creating it.
 - c. Optionally, in the **Description** field, enter a description of the ESB system or service group.
 - d. Click **OK**.
4. In the Application Navigator, click the refresh icon.

6.4.2 What Happens When You Create an ESB System or Service Group

When an ESB system is created, an entry for the system in the form *system_name.esbsys*, is added to the Resources folder under the project name in the Application navigator. If an ESB service group is created, the file has the extension *esbgrp*.

The ESB system file contains information for this system. You can double-click on the entry in the Application navigator or right-click on the entry and select Open from the menu to view and edit the source of the ESB system file. You can also update the system definition in Oracle ESB Control. See [Section 12.3, "Using the Services View to Review and Update ESB Systems"](#).

[Example 6–1](#) shows sample source code for an ESB system file.

Example 6–1 Sample ESB System File

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<system name="Fulfillment" guid="3C2D63E0CC1811DABFA0CDAACB57527A" qname="Fulfillment"
status="ENABLED">
  <versionInfo>
    <ID>1153839756546</ID>
    <lastModifiedTime>Tue Jul 25 08:02:36 PDT 2006</lastModifiedTime>
    <lastModifiedUser>JDev_SOADEMO_FulfillmentESB</lastModifiedUser>
  </versionInfo>
  <description/>
  <parameters>
    <parameter name="Fax"/>
    <parameter name="Mobile"/>
    <parameter name="Host">localhost</parameter>
    <parameter name="Phone"/>
    <parameter name="Pager"/>
    <parameter name="Email"/>
    <parameter name="Port">8888</parameter>
  </parameters>
  <clusterName>esb</clusterName>
  <deferredQueueName numberOfListeners="1"
topicConnectionFactory="">OracleASjms/ESBDeferredTopic</deferredQueueName>
</system>
```

6.5 Browsing for Deployed Services Using the Service Explorer Browser

You can browse for deployed services with the Service Explorer. The Service Explorer is available when creating a routing service or an adapter service. For an example of creating an adapter service, see [Section 6.8.1, "How to Add Adapter Services"](#) and [Section 6.9.1, "How to Add Routing Services"](#).

For information about browsing for deployed services in Oracle Enterprise Service Bus, see "Creating Inbound and Outbound Services" in *Oracle Enterprise Service Bus Developer's Guide*.

6.5.1 How to Browse for Deployed Services

You browse for deployed services by launching the Service Explorer from a create service dialog.

To browse for deployed services:

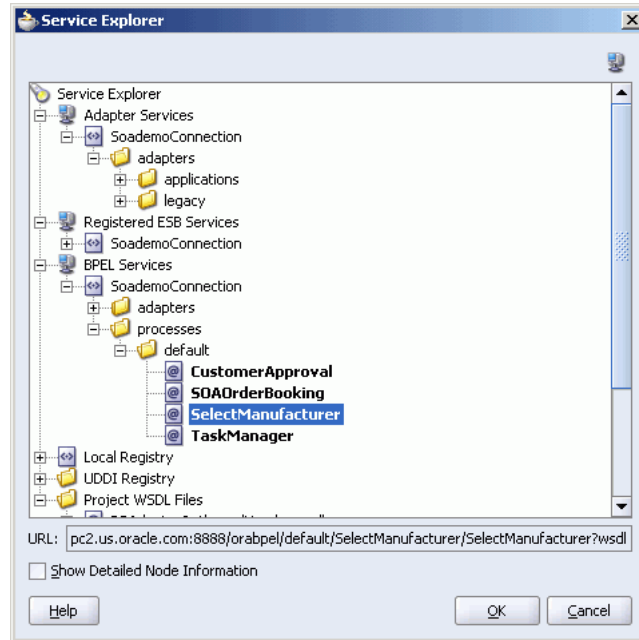
1. If you have not already done so, open the Create Adapter, Routing, or SOAP Service dialog.
2. Enter information for other fields on the page, such as Name, System/Group, and Description.
3. If you are creating a routing service, choose **Select Existing WSDL** in the Create Routing Service dialog.
4. To the right of the **WSDL Location** field, click the flashlight (torch) icon to open the Service Explorer dialog.
5. Select a branch of the Service Explorer tree, such as **BPEL Server Connection**.
6. Expand the server connection folder to view the existing projects.

For example, expand the integration server connection that is set up.

7. Expand the project to view the deployed services.
8. Select a deployed service.

Figure 6–3 is an example of a Service Explorer dialog with a service selected.

Figure 6–3 Service Explorer Dialog



9. Click **OK**.

The Service Explorer dialog closes and you are returned to the Create Routing Service, SOAP Service, or Adapter Service dialog.

10. In the Port Type field, click the **down arrow**, and then select the port type for the routing service.
11. Click **OK**.

6.6 Adding Endpoint Properties for Adapter or SOAP Services

Note: Endpoint properties are not fully tested and have been marked as a *Preview* feature. Preview features will be supported in a later release. To start JDeveloper in Preview mode, run the following at an operating system prompt:

```
JDeveloper_Home/jdev/bin/jdev.exe -J"-Dpreview_mode=true"
```

Each adapter type and SOAP service has a specific set of endpoint properties that are provided when the service is created. For example, you can specify a new URL, or URLs, for the Location endpoint property of a SOAP service. Or you can also change the RetryCount and RetryInterval endpoint properties to override the default values for adapter error handling.

There are two types of endpoint properties: predefined and logical.

- **Predefined (override) properties**

These properties are defined in the adapter wizard and the value can be overridden or a value can be provided at runtime.

- **Logical properties**

For the file and FTP adapters you can specify that the directories are logical rather than physical. These logical names appear as endpoint properties in the Endpoint Property Chooser dialog.

In addition, any adapter property can be made into a logical property by inserting a \$ into the wizard value (or manually edit the WSDL file). For example, if the user specifies \$myName as a value in the adapter wizard, then myName appear in the Endpoint Property Chooser dialog.

You can add, delete, update, or view endpoint properties for the service in the Endpoint Properties panel when modifying an adapter or SOAP service in Oracle JDeveloper ESB Designer. See [Section 6.7.2, "How to Modify SOAP Services"](#) and [Section 6.8.2, "How to Modify Adapter Services"](#). You can also access endpoints in the Property tab of the Oracle ESB Control Services view. See [Section 12.4, "Using the Services View to Review and Update ESB Services"](#).

6.6.1 How to Add Endpoint Properties

You can modify endpoint properties in Oracle JDeveloper ESB Designer by opening the adapter or SOAP service property page.

To add endpoint properties:

1. In the **Design** tab, double click on the upper section of the adapter or SOAP service icon. The cursor is shaped like a hand on that region of the icon. An example of the Design tab is shown in [Figure 6-7](#).
2. In the Endpoint Properties panel, you can add, delete, update, or view endpoint properties for the service.

Click the + next to **Endpoint Properties** to open the panel. The Endpoint Properties panel includes the following options:

- **Large Green Plus Sign (+) in upper right**

Opens the Endpoint Properties Chooser dialog to enable you to add an endpoint property to the service. Select a property from the list in the Endpoint Properties Chooser dialog and click the OK button to add the endpoint property. Place the cursor over the property name to view the tooltip description for the property.

- **Large Red X in upper right**

Deletes the selected endpoint property.

- **Name**

Name of the endpoint property. Place the cursor over the property name to view the tooltip description for the property.

- **Value**

Click in this field to enter or update the value of the endpoint property.

3. Select **File > Save** to save your changes to the property page.

6.7 Adding SOAP Services to Oracle Enterprise Service Bus

Services are the means by which data gets from one application to another and in and out of the Oracle Enterprise Service Bus environment. Connectivity is provided through adapter services and Simple Object Access Protocol (SOAP) invocation services. For example, an ESB SOAP service can be used as an outbound adapter to move data to a PeopleSoft or Siebel application.

Data contained within XML messages are distributed from the source application to a target application using routing services. As the name suggests, a routing service determines how a message gets from one point to another within the Oracle Enterprise Service Bus environment. ESB includes a standards-based data mapper. The data mapper specifies an xsl file to transform data from one XML schema to another, thus enabling data interchange among applications using different schemas.

For information about managing SOAP services in Oracle Enterprise Service Bus, see "Creating Inbound and Outbound Services" in *Oracle Enterprise Service Bus Developer's Guide*.

6.7.1 How to Add SOAP Services

To create an ESB SOAP service, you drag a SOAP service type from the Components palette and drop it onto the Design tab. When you do so, a wizard or dialog opens to assist you in creating the service.

To create a SOAP service with Oracle JDeveloper ESB Designer:

1. In the Application Navigator, navigate to the ESB project for which you want to create a SOAP service, expand the **Resources** folder. Double click *project-name.esb*, where *project-name* is the name of the project to which you want to add the SOAP service.

The Design tab for the project is displayed.

2. In the **Component Palette**, click the down arrow and select ESB services if not already selected.
3. Drag and drop **SOAP Service** into the **Design** tab.

The Create SOAP Invocation Service dialog opens.

4. In the Create SOAP Invocation Service dialog, enter the **Name**, **System/Group**, and an optional **Description** for the service.

Name: Enter a unique name across the ESB system in which you are creating the service; spaces are not allowed.

System/Group: Click on the flashlight (torch) to open the ESB Service Group Browser dialog and select the system/group for this service.

Description: Enter an optional description for this service.

5. To complete WSDL File field, you can click on the following icons to the right of the field:
 - Browse WSDL Files icon

Click on this icon to open a dialog that enables you to search for and open a WSDL file from the local file system.

- Service Explorer icon

Click on this icon to launch the Service Explorer and search for and select a WSDL file of an existing service.

For information on using the Service Explorer, see [Section 6.5.1, "How to Browse for Deployed Services"](#).

6. In the **Port Type** field, click the **down arrow**, and then select the port type for the service.
7. Click **OK** when finished.

The new SOAP service appears in the Design tab.

6.7.2 How to Modify SOAP Services

You can modify SOAP services in Oracle JDeveloper ESB Designer by opening the SOAP service property page.

To modify a SOAP service:

1. In the **Design** tab, double click on the upper section of the SOAP service icon. The cursor is shaped like a hand on that region of the icon. An example of the Design tab is shown in [Figure 6-7](#).
2. Make your changes to the adapter service information that can be modified. Note that the name of the service cannot be changed.
3. In the Endpoint Properties panel, you can add, delete, update, or view endpoint properties for the service. For information about endpoint properties, see [Section 6.6, "Adding Endpoint Properties for Adapter or SOAP Services"](#).
4. Save your changes.

6.7.3 How to Delete SOAP Services

You can delete a SOAP service in the Oracle JDeveloper ESB Designer Design tab.

To delete a SOAP service:

1. In the **Design** tab, select the SOAP service icon.
2. Click the large red **X** at the top of Design tab to delete the selected service.
3. Confirm that you want to delete the selected service.
4. Save your changes.

Note: Do not delete SOAP services in the Application Navigator.

6.7.4 What Happens When You Add SOAP Services

When you complete the information for adding a new SOAP service, the service icon displays in the Design tab and the service definition file appears in the Application Navigator in the Resources folder of the ESB project. The name of the service entry in the Application Navigator includes the ESB system name, the service name, and the `esbsvc` extension, in the form `system-name_service-name.esbsvc`. The actual source of the service definition file is not visible in Oracle JDeveloper ESB Designer.

When the ESB project is registered with the ESB server, the ESB files for the SOAP service that are created in Oracle JDeveloper ESB Designer are deployed to the design time metadata server. See [Section 6.2.1, "What Happens When you Create an Oracle Enterprise Service Bus Flow"](#).

6.7.5 What Happens at Runtime

During runtime, a SOAP service receives a message instance from an external source and passes it to an ESB routing service for processing or an ESB routing service passes a message instance to a SOAP service which sends it to an external source.

6.8 Adding Adapter Services to Oracle Enterprise Service Bus

Oracle Enterprise Service Bus provides support for creating and modifying services for the Oracle Technology adapters. The Oracle Technology adapters enable you to integrate mainframe and legacy applications with enterprise resource planning (ERP), customer relationship management (CRM), database, and messaging systems.

[Table 6–2](#) provides a summary of the Oracle Technology adapter services you can create. Adapters services can be configured as inbound or outbound adapters services. Inbound adapter services send messages to the enterprise service bus, while outbound adapter services send messages to an application or system external to the enterprise service bus.

Table 6–2 Summary of Oracle Technology Adapters

Adapter Service	Description
AQ Adapter Service	Sends or receives messages from Oracle Advanced Queuing single or multiconsumer queues
Database Adapter Service	Sends or receives messages extracted from an Oracle Database table or created by executing a stored procedure
File Adapter Service	Sends or receives messages from a file in the local file system
FTP Adapter Service	Sends or receives messages from a file at a remote FTP server
JMS Adapter Service	Sends or receives messages from a JMS queue or topic
MQ Adapter	Sends or receives messages from IBM's MQ Series
Oracle Applications Adapter Service	Sends or receives messages from an Oracle E-Business Suite interface

Refer to the following resources for more information:

- For an overview of adapters, see *Oracle Application Server Adapter Concepts*.
- For specific details about configuring adapters for Oracle Enterprise Service Bus, see *Oracle Application Server Adapter for Files, FTP, Databases, and Enterprise Messaging User's Guide*.
- For information about using the Oracle Applications adapter for Oracle E-Business Suite, see *Oracle Application Server Adapter for Oracle Applications User's Guide*.
- For information about managing adapters in Oracle Enterprise Service Bus, see "Creating Inbound and Outbound Services" in *Oracle Enterprise Service Bus Developer's Guide*.
- For a tutorial that describes how to design an Oracle Enterprise Service Bus flow that uses adapter services, see *Oracle Enterprise Service Bus Quick Start Guide*.

6.8.1 How to Add Adapter Services

Oracle JDeveloper ESB Designer provides wizards that assist you in creating inbound and outbound adapter services. The wizard collects the necessary information to generate the WSDL file that defines the service.

To add an adapter service:

1. In the Application Navigator, navigate to the ESB project for which you want to create a SOAP service, expand the **Resources** folder. Double click *project-name.esb*, where *project-name* is the name of the project to which you want to add the SOAP service.

The Design tab for the project is displayed.

2. In the **Component Palette**, click the down arrow and select **Adapter Services** if not already selected.
3. Drag and drop an adapter into the Design tab.

The create adapter service dialog for the specific adapter type opens.

4. Enter the **Name**, **System/Group**, and an optional **Description** for the adapter.

Name: Enter a unique name across the ESB system in which you are creating the service; spaces are not allowed.

System/Group: Click on the flashlight (torch) to open the ESB Service Group Browser dialog and select the system/group for this service. For example:

Fulfillment under **Systems/Groups in project**.

Description: Enter an optional description.

5. To complete WSDL File field, you can click on the following icons to the right of the field:

- Configure adapter service wsdl icon

Click on this icon to launch the adapter configuration wizard for the specific type of adapter you have selected from the Component Palette. The wizard guides you through the setups.

For details about configuring the AQ, database, file, FTP, JMS, and MQ adapters in Oracle JDeveloper ESB Designer, see *Oracle Application Server Adapter for Files, FTP, Databases, and Enterprise Messaging User's Guide*. For details about configuring the Oracle application adapter for Oracle E-Business Suite, see *Oracle Application Server Adapter for Oracle Applications User's Guide*.

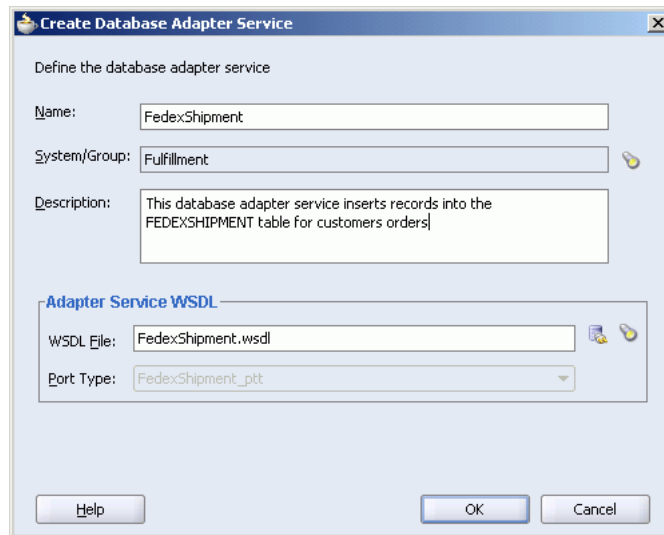
- Service Explorer icon

Click on this icon to launch the Service Explorer and search for deployed services.

See [Section 6.5.1, "How to Browse for Deployed Services"](#).

6. Click **OK**.

[Figure 6–4](#) is an example of a completed Create Adapter Service dialog.

Figure 6–4 Create Adapter Service Dialog

The new adapter service appears in the Design tab, as shown in [Figure 6–1](#). The adapter service requires a routing service to route message instances to or from the adapter service.

6.8.2 How to Modify Adapter Services

You can modify adapter services in Oracle JDeveloper ESB Designer by opening the adapter property page.

To modify an adapter service:

1. In the **Design** tab, double click on the upper section of the adapter service icon. The cursor is shaped like a hand on that region of the icon. An example of the Design tab is shown in [Figure 6–7](#).
2. Make your changes to the adapter service information that can be modified. Note that the name of the service cannot be changed.
3. In the Endpoint Properties panel, you can add, delete, update, or view endpoint properties for the service. For information about endpoint properties, see [Section 6.6, "Adding Endpoint Properties for Adapter or SOAP Services"](#).
4. Save your changes.

6.8.3 How to Delete Adapter Services

You can delete an adapter service in the Oracle JDeveloper ESB Designer Design tab.

To delete an adapter service:

1. In the **Diagram** tab, select the adapter service icon.
2. Click the large red **X** at the top of Design tab to delete the selected adapter service.
3. Confirm that you want to delete the selected service.
4. Save your changes.

Note: Do not delete adapter services in the Application Navigator.

6.8.4 What Happens When You Add Adapter Services

When you complete the information for adding a new adapter service, the adapter service icon displays in the Design tab and the service definition file appears in the Application Navigator in the Resources folder of the ESB project. The name of the adapter entry in the Application Navigator includes the ESB system name, the adapter name, and the esbsvc extension, in the form *system-name_service-name.esbsvc*. The actual source of the service definition file is not visible in Oracle JDeveloper ESB Designer.

A WSDL file is created for the service of the form *service_name.wsdl* which appears in the Application Navigator. The WSDL file defines the input and output messages for this instance flow, the supported client interface and operations, and other features. [Example 6-2](#) is a sample adapter WSDL file.

Example 6-2 Sample Adapter WSDL File

```
<definitions
  name="USPSShipment"
  targetNamespace="http://xmlns.oracle.com/pcbpel/adapter/file/USPSShipment/"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:tns="http://xmlns.oracle.com/pcbpel/adapter/file/USPSShipment/"
  xmlns:plt="http://schemas.xmlsoap.org/ws/2003/05/partner-link/"
  xmlns:jca="http://xmlns.oracle.com/pcbpel/wsdl/jca/"
  xmlns:impl="http://www.globalcompany.com/ns/USPSShipment"
  xmlns:hdr="http://xmlns.oracle.com/pcbpel/adapter/file/"
>
  <import namespace="http://xmlns.oracle.com/pcbpel/adapter/file/"
location="fileAdapterOutboundHeader.wsdl"/>
  <types>
    <schema xmlns="http://www.w3.org/2001/XMLSchema" >
      <import namespace="http://www.globalcompany.com/ns/USPSShipment"
schemaLocation="USPSShipment.xsd" />
    </schema>
  </types>
  <message name="shipment_msg">
    <part name="shipment" element="impl:shipment"/>
  </message>
  <portType name="Write_ptt">
    <operation name="Write">
      <input message="tns:shipment_msg"/>
    </operation>
  </portType>
  <binding name="Write_binding" type="tns:Write_ptt">
    <jca:binding />
    <operation name="Write">
      <jca:operation
        PhysicalDirectory="C:\SOAFilesOut"
        InteractionSpec="oracle.tip.adapter.file.outbound.FileInteractionSpec"
        FileNamingConvention="shipment_%SEQ%.txt"
        NumberMessages="1"
        OpaqueSchema="false" >
      </jca:operation>
    <input>
      <jca:header message="hdr:OutboundHeader_msg" part="outboundHeader"/>
    </input>
  </operation>
</binding>
```

```

<service name="USPSShipment">
  <port name="Write_pt" binding="tns:Write_binding">
    <jca:address location="eis/FileAdapter" />
  </port>
</service>
<plt:partnerLinkType name="Write_plt" >
  <plt:role name="Write_role" >
    <plt:portType name="tns:Write_ptt" />
  </plt:role>
</plt:partnerLinkType>
</definitions>

```

A standard header file is created for the adapter service of the form *adapter_typeAdapterOutboundHeader.wsdl* or *adapter_typeAdapterInboundHeader.wsdl*, depending on whether the service is inbound or outbound. The *adapter_type* specifies the type of adapter, such as DB, file, or jms. [Example 6-2](#) is a sample adapter header file.

Example 6-3 Sample Adapter Header File

```

definitions
  name="fileAdapter"
  targetNamespace="http://xmlns.oracle.com/pcbpel/adapter/file/"
  xmlns:tns="http://xmlns.oracle.com/pcbpel/adapter/file/"
  xmlns="http://schemas.xmlsoap.org/wsdl/" >
  <types>
    <schema attributeFormDefault="qualified" elementFormDefault="qualified"
      targetNamespace="http://xmlns.oracle.com/pcbpel/adapter/file/"
      xmlns="http://www.w3.org/2001/XMLSchema"
      xmlns:FILEAPP="http://xmlns.oracle.com/pcbpel/adapter/file/">
      <element name="OutboundFileHeaderType">
        <complexType>
          <sequence>
            <element name="fileName" type="string"/>
          </sequence>
        </complexType>
      </element>
    </schema>
  </types>
  <!-- Header Message -->
  <message name="OutboundHeader_msg">
    <part element="tns:OutboundFileHeaderType" name="outboundHeader"/>
  </message>
</definitions>

```

When the ESB project is registered with the ESB server, the ESB files for the adapter service that are created in Oracle JDeveloper ESB Designer are deployed to the design time metadata server. See [Section 6.2.1, "What Happens When you Create an Oracle Enterprise Service Bus Flow"](#).

6.8.5 What Happens at Runtime

During runtime, an inbound adapter service receives a message instance from an external source and sends the message to an ESB routing service for processing.

An outbound adapter receives a message instance from an ESB routing service and sends the message to an external source.

6.9 Adding Routing Services to Oracle Enterprise Service Bus

A routing service is the key component for moving a message across the enterprise service bus – from its entry point to its exit point. Oracle JDeveloper ESB Designer provides tools that assist you in creating a routing service

The following are the key components that define a routing service:

- WSDL file
- Target services and operations
- Transformation definition
- Filter Expression
- Execution type (synchronous or asynchronous)
- The ESB systems from which the routing service accepts messages

The WSDL specifies how other services (either within or outside of the enterprise service bus) call the routing service. The remaining items, referred to as routing rules, determine where the routing service sends each message it receives, how it sends it, and what, if any, changes it makes to the message structure prior to sending it to the target service.

Oracle JDeveloper ESB Designer provides tools that assist you in creating the routing service WSDL and defining the routing rules.

There are two main ways you can create the routing service WSDL:

- By specifying an existing WSDL file and one of the port types defined within that WSDL

This option enables you to use an existing WSDL file (on the local file system or at an Oracle JDeveloper ESB Designer connection) to define the routing service. Oracle JDeveloper ESB Designer parses the WSDL you specify to present the list of port types from which you make a selection.

If you want to edit an existing WSDL for the routing service, edit it using a WSDL editor (such as Oracle JDeveloper) prior to specifying it in this dialog.

- By generating the WSDL from a schema file (.xsd)

This option enables you to use an existing XSD file or a file in a native file format (such as a comma-separated value (CSV) file, a fixed-length file, a document type definition (DTD) file or a COBOL copybook file) to define the routing service.

You can specify the same or different schema files for the request, response, and fault message schemas, which Oracle JDeveloper ESB Designer converts into WSDL input, output, and fault elements in the WSDL. Minimally, you must specify the schema for the request message. You cannot specify a fault message schema, unless you also specify a response.

In addition, you specify the operation and namespace. Oracle JDeveloper ESB Designer converts the operation into a operation element in the WSDL file and the namespace you specify is defined as the `tns` namespace in the WSDL file.

The operation element describes how input to the operation is defined.

For information about managing routing services in Oracle Enterprise Service Bus, see "Creating Routing Services and Routing Rules" in *Oracle Enterprise Service Bus Developer's Guide*.

6.9.1 How to Add Routing Services

To create a routing service, you complete the Create Routing Service dialog available in Oracle JDeveloper ESB Designer. You can access this dialog using either of the following two methods described here.

To add an adapter service schematically using the Design tab:

1. In the Application Navigator, navigate to the ESB project for which you want to create a routing service, expand the **Resources** folder and double click *project-name.esb*, where *project-name* is the name of the project to which you want to add the routing service.

The Design tab for the project is displayed.

2. In the **Component Palette**, click the down arrow and select ESB services if they are not already selected.
3. Drag and drop **Routing Service** into the **Diagram** tab.

The Create Routing Services dialog opens.

4. You can also right-click in the Design tab, then select **Create ESB Service > Routing Service**.

To create the routing service using dialog boxes only:

1. In the Applications navigator, right-click the ESB project for which you want to create a routing service, and then click **New**.

The New Gallery opens.

2. In the **Categories** panel, expand **Business Tier**, and then click **Web Services**.
3. In the **Items** panel, click **ESB Routing Service**, and then click **OK**.

The Create Routing Service dialog opens.

The Create Routing Service dialog provides three methods for specifying the WSDL for the routing service, as described in the following sections. Each of these sections provides step-by-step instructions on completing the Create Routing Service dialog for the method you want to use.

[Figure 6–5](#) is an example of a Create Routing Service dialog with the Request tab displayed.

Figure 6–5 Create Routing Service - Request Tab

6.9.1.1 How to Generate the WSDL for a Routing Service from an Existing XSD File

Use this method to provide an existing WSDL for the routing service. After you specify the file, Oracle JDeveloper ESB Designer parses it to determine the defined schema elements and presents them in a drop-down list from which you can make a selection.

To generate a WSDL from an existing XSD file:

1. If you have not already done so, open the Create Routing Service dialog.
2. In the **Name** field, enter a name for the routing service.
The name must be unique within the scope of the project in which the routing service is being created. Spaces are not allowed.
3. For the **System/Group** field, click **Browse** to open the ESB Service Group Browser dialog, select the system (and service group, if desired) to which you want to add the routing service, and then click **OK**.
To create a new system or service group to contain the routing service you are creating, click **Create New** at the top of the ESB Service Group Browser dialog. See [Section 6.4.1, "How to Create an ESB System or Service Group"](#) for information on creating a new ESB system or service group.
4. In the **Description** field, enter a description for the routing service, if desired. This field is optional.
5. Choose **Generate WSDL from Schemas**.
This option includes the **Request**, **Reply**, and **Fault** tabs.

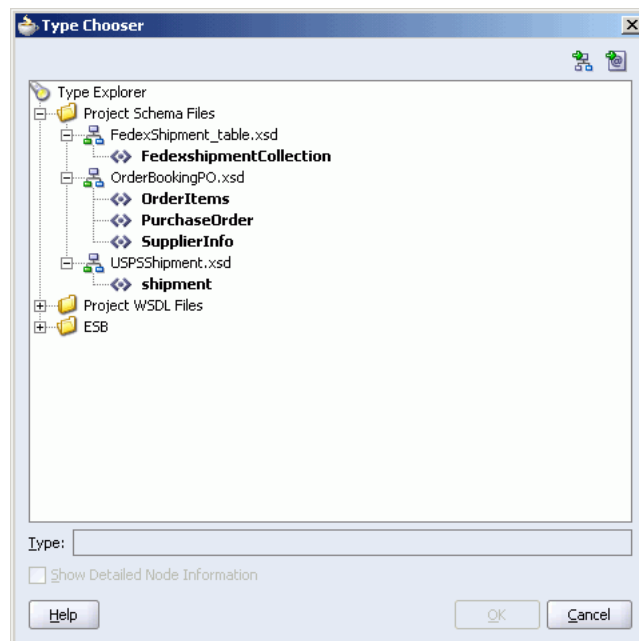
6. On the **Request** tab, click **Browse** to access the **Schema Location**.

The Type Chooser dialog opens and presents the schema files (XSD files) from which you can choose to generate the WSDL. Expand the trees under **Project Schema Files**, **Project WSDL Files**, and **ESB** to locate the schema. Navigate to the root element of the XSD file for the message instance that you want this routing service to process. Select the element and click OK.

You can also import a schema or WSDL file by clicking on the Import Schema File or Import WSDL icon that appears at the top right of the Type Chooser dialog.

Figure 6–6 is an example of a Type Chooser dialog with the Import Schema Files tree expanded.

Figure 6–6 Type Chooser Dialog



7. In the **Request** tab **Schema Element** field, select the root element for the message that you want this routing service to process if not already selected.
8. On the **Reply** tab, repeat the steps for the **Request** tab if entering any information.
9. On the **Fault** tab, repeat the steps for the **Request** tab if entering any information.
10. In the **Operation Name** field, enter the operation name. Spaces are not allowed.
11. In the **Namespace** field, enter a namespace or accept the current value.
12. Click **OK**.

The routing service is created and an icon is added to the ESB Project's Design tab.

6.9.1.2 How to Generate the WSDL to Create a Routing Service Based on a Sample File

Oracle JDeveloper ESB Designer provides a wizard that assists you in creating the XSD representation of various file formats (such as CSV file, fixed-length file, DTD, and Cobol copybooks) based on a sample file and details that you provide about the file's structure. You can then direct Oracle JDeveloper ESB Designer to generate the WSDL for the routing service from that XSD file.

To generate a WSDL based on a sample file:

1. If you have not already done so, open the Create Routing Service dialog.
2. In the **Service Name** field, enter a name for the routing service.
The name must be unique within the scope of the project in which the routing service is being created. Spaces are not allowed.
3. For the **System/Group** field, click **Browse** to open the ESB Service Group Browser dialog, select the system (and service group, if desired) to which you want to add the routing service, and then click **OK**.
Click **Help** for assistance in using the ESB Service Group Browser dialog.
4. In the **Description** field, enter a description for the routing service, if desired. This field is optional.
5. Choose **Generate WSDL from Schema**.
This option includes the **Request**, **Reply**, and **Fault** tabs.
6. On the **Request** tab, click **Define Schema for Native Format**.
The Native Format File Builder wizard opens.
7. Follow the steps through the wizard.
If you need assistance on a wizard page, click **Help**.
8. In the **Schema Element** field on the **Request** tab, select the root element for the message that you want this routing service to process.
9. **Reply** tab
10. **Fault** tab
11. **Operation Name**
Spaces are not allowed.
12. **Namespace**
13. Click **OK**.

The routing service is created and an icon is added to the ESB project's Design tab.

6.9.1.3 How to Select an Existing WSDL to Create a Routing Service

If you use this method to provide the WSDL for the routing service, the existing WSDL must exist on the local file system. After you specify the file, Oracle JDeveloper ESB Designer parses it to determine the defined port types and presents them in a drop-down list from which you can make a selection.

To select an existing WSDL:

1. If you have not already done so, open the Create Routing Service dialog.
In the **Service Name** field, enter a name for the routing service.
The name must be unique within the scope of the project in which the routing service is being created. Spaces are not allowed.
2. For the **System/Group** field, click **Browse** to open the ESB Service Group Browser dialog, select the system (and service group, if desired) to which you want to add the routing service, and then click **OK**.
Click **Help** for assistance in using the ESB Service Group Browser dialog.

3. In the **Description** field, enter a description for the routing service, if desired. This field is optional.
4. Choose **Select Existing WSDL**.
5. To complete WSDL File field, you can click on the following icons to the right of the **WSDL File** field:
 - Browse WSDL Files
Click on this icon to launch the **Open** files dialog. In the Open dialog, navigate to an existing WSDL file, and then click **Open** to close the dialog. If you need assistance with this dialog, click **Help**.
 - Service Explorer icon
Click on this icon to launch the Service Explorer and search for deployed services. See [Section 6.5.1, "How to Browse for Deployed Services"](#). Click **OK** when done to close the dialog.
6. In the Port Type field, click the **down arrow**, and then select the port type for the routing service.
7. Click **OK**.

The routing service is created and an icon is added to the ESB project's Design tab.

6.9.2 How to Modify Routing Services

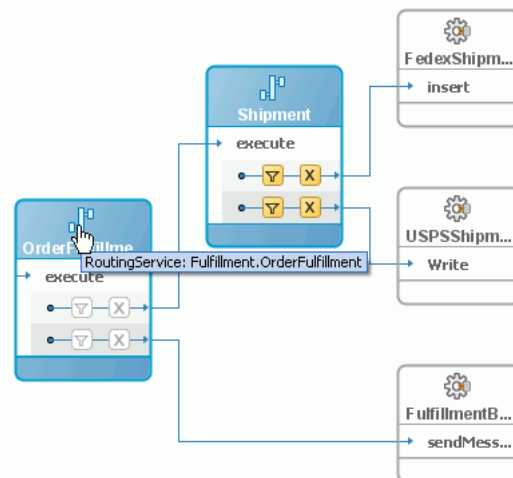
You can modify a routing service in Oracle JDeveloper ESB Designer.

To modify a routing service:

1. In the **Design** tab, double-click on the upper section of the routing service icon. The cursor is shaped like a hand on that region of the icon.

[Figure 6–7](#) is an example of a ESB Project Design tab with service icons.

Figure 6–7 Design Tab - Service Icons



2. Update the properties of the routing service that can be modified. Note that the name of the service, WSDL file, and port type cannot be changed.

You can also update the routing rules on this property page. See [Section 6.10, "Specifying Routing Rules"](#).

3. Save your changes.

6.9.3 How to Delete Routing Service

You can delete a routing service in Oracle JDeveloper ESB Designer.

To delete a routing service:

1. In the **Design** tab, select the routing service icon.
2. Click the large red X at the top of Design tab to delete the selected routing service.
3. Confirm that you want to delete the selected service.
4. Save your changes.

Note: Do not delete routing services in the Application Navigator.

6.9.4 What Happens When You Add Routing Services

When you complete the information for adding a new routing service, the routing service icon displays in the Design tab and the service definition file appears in the Application Navigator in the Resources folder of the ESB project. The name of the routing service entry in the Application Navigator includes the ESB system name, the routing service name, and the esbsvc extension, in the form *system-name_service-name.esbsvc*. The actual source of the service definition file is not visible in Oracle JDeveloper ESB Designer.

A WSDL file is created for the service of the form *service_name.wsdl* which appears in the Application Navigator. [Example 6–4](#) is a sample routing service WSDL file.

Example 6–4 Sample Routing Service WSDL File

```
<?xml version= '1.0' encoding= 'UTF-8' ?>
<definitions
  name="OrderFulfillment"
  targetNamespace="http://www.globalcompany.com/ns/Fulfillment"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:tns="http://www.globalcompany.com/ns/Fulfillment"
  xmlns:inp1="http://www.globalcompany.com/ns/order">
  <types>
    <schema xmlns="http://www.w3.org/2001/XMLSchema" >
      <import namespace="http://www.globalcompany.com/ns/order"
        schemaLocation="OrderBookingPO.xsd" />
    </schema>
  </types>
  <message name="PurchaseOrder_request">
    <part name="PurchaseOrder" element="inp1:PurchaseOrder"/>
  </message>
  <portType name="execute_ppt">
    <operation name="execute">
      <input message="tns:PurchaseOrder_request"/>
    </operation>
  </portType>
</definitions>
```

When the ESB project is registered with the ESB server, the ESB files for the routing service that are created in Oracle JDeveloper ESB Designer are deployed to the design time metadata server. See [Section 6.2.1, "What Happens When you Create an Oracle Enterprise Service Bus Flow"](#).

6.9.5 What Happens at Runtime

During runtime, a routing service sends or receives a message instance to or from an adapter, routing, or SOAP service for processing. The routing rules determines the conditions of the routing and the data transformation XSL file maps the data from a source to a target.

6.10 Specifying Routing Rules

After you define a routing service, by specifying its WSDL, you can specify the rules that determine how a message instance processed by the routing service gets to its next destination. Routing rules can be defined using a property panel in Oracle JDeveloper ESB Designer or a property sheet in Oracle ESB Control.

You must specify at least one target operation and you can specify multiple target operations. For each target operation, you can apply a data transformation, set up filtering, specify from which system messages are accepted, specify asynchronous or synchronous execution, and establish priority when there are multiple target operations.

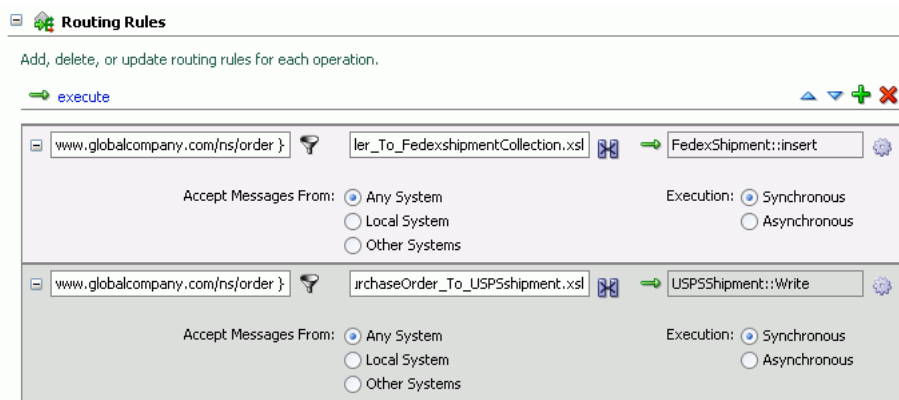
6.10.1 How to Access Routing Rules

To access the Routing Rules property panel in Oracle JDeveloper ESB Designer, use either of the following two methods described here.

To access the routing rules properties panel:

- From the Applications Navigator:
 - a. In the Application Navigator, expand the **Resources** folder under the ESB project.
 - b. In the **Resources** folder, double click the name of the routing service for which you want to specify routing rules.
 - c. Click the plus symbol (+) to expand the **Routing Rules** information.
- From the Design tab:
 - a. Double-click the icon that represents the routing service for which you want to specify routing rules.
 - b. Click the plus symbol (+) to expand the **Routing Rules** information.

[Figure 6–8](#) is an example of a routing rules panel with multiple rules specified.

Figure 6–8 Routing Rules Panel

6.10.2 How to Specify Target Operations

The target operation is the only routing rule you must specify to make use of a routing service. This routing rule tells the routing service the next service, known as the target service, to which the message should be sent and the operation to perform on that message when it reaches the target service.

You can specify multiple target service and target operation pairs for each routing service.

In addition, for request/reply message flows, you can forward the reply message to another target, and specify a target service in the event that a message fault occurs. The following are the configurations that you can specify a target operation:

- Specifying Target Operations for a One-Way Configuration
- Specifying Target Operations to Return a Response to the Source Service
- Specifying Target Operations to Forward a Response to a Non-Source Service
- Specifying Target Operations for a Faulted Message

To Specify Target Operations for a Routing Rule

1. If you have not already done so, expand the Routing Rules information panel.
2. Click the large green **plus (+)** button.

The Browse Target Service Operation dialog opens.

3. In the Browse Target Service Operation dialog, follow these steps:
 - a. Navigate to, and then expand the desired target service.
 - b. Select the target service operation.
 - c. Click **OK**.

The Routing Rules tab is updated to reflect the newly added target service and operation, with the value expressed as *target_service.target_operation*.

4. Repeat steps two and three if you want to add an additional target service and target operation pair. You can specify the same target service and a different operation, if desired.

Note: To modify the target service and target service operation, click the Browse for Target Service Operation icon.

6.10.3 How to Specify the ESB Systems From which Messages are Accepted

You can indicate the ESB systems from which the routing service will accept messages.

To specify from which system the service will accept messages:

1. If you have not already done so, expand the Routing Rules information, as described in [Section 6.10.1, "How to Access Routing Rules"](#).
2. Choose one of the following options:
 - Any System
Choose this option to specify that the routing service accept messages that originate from any ESB system, including the one in which the routing service was created.
 - Same System
Choose this option to specify that the routing service accept messages that originate only from the same ESB system as the one in which the routing service was created.
 - Other Systems
Choose this option to specify that the routing service accept messages that originate only from ESB systems other than the one in which the routing service was created.

6.10.4 How to Specify Synchronous or Asynchronous Execution

Synchronous execution provides an immediate response to a request; asynchronous does not. Asynchronous execution is useful for environments in which a service can take a long time to process a request. Asynchronous services also provide a more reliable fault-tolerant and scalable architecture than synchronous services.

To specify synchronous or asynchronous execution:

1. If you have not already done so, expand the Routing Rules information, as described in [Section 6.10.1, "How to Access Routing Rules"](#).
2. Click the **plus (+)** button to expand the routing rules information.
The **Execution** options display on the right side of the property sheet.
3. Specify **Synchronous** or **Asynchronous** execution with the appropriate button.

6.10.5 How to Specify Routing Rules Priority

You can choose the priority of multiple routing rules of a routing service by placing the rules in ascending order, with the top rules having the highest priority.

To specify priority of routing rules:

1. If you have not already done so, expand the Routing Rules information, as described in [Section 6.10.1, "How to Access Routing Rules"](#).
2. Click the **plus (+)** button to select and expand the routing rules information.
3. Click the up or down triangles in the upper right of the routing rules panel to move the selected rule to order of the correct priority. For an illustration of the property sheet, see [Figure 6-8](#).

6.10.6 How to Create an XSL File for Data Structure Transformation

In a routing rule, you can specify an XSL file to transform data from one XML schema to another using the XSL Data Mapper tool. This enables data interchange among applications using different schemas. For example, you can map incoming source purchase order schema to an outgoing invoice schema.

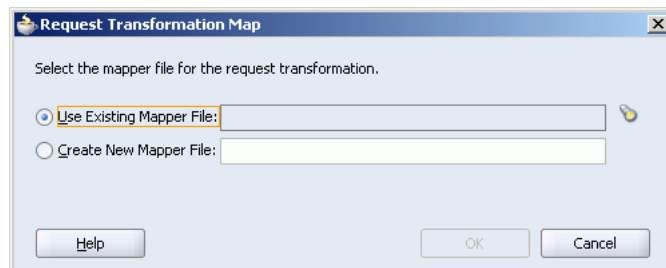
After you define an XSL file, you can reuse it in multiple routing rule specifications.

To create an XSL file for data transformation:

1. If you have not already done so, open the Routing Rules information, as described in [Section 6.10.1, "How to Access Routing Rules"](#).
2. Click the **Specify Transformation** icon. The icon to the right of the field in the middle of the routing rules panel, as shown in [Figure 6-8](#).

The Request Transformation Map dialog opens, as shown in [Figure 6-9](#).

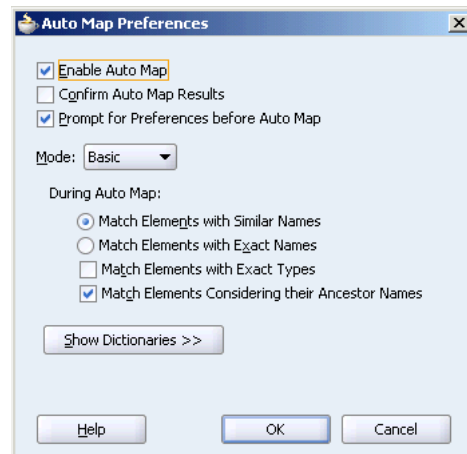
Figure 6-9 Request Transformation Map Dialog



3. Choose one of the following options:
 - **Create New Mapper File** and then enter a name for the file (or accept the default value).
 - **Use Existing Mapper File** and then click the flashlight icon to browse to the existing mapper file
4. Click **OK**.

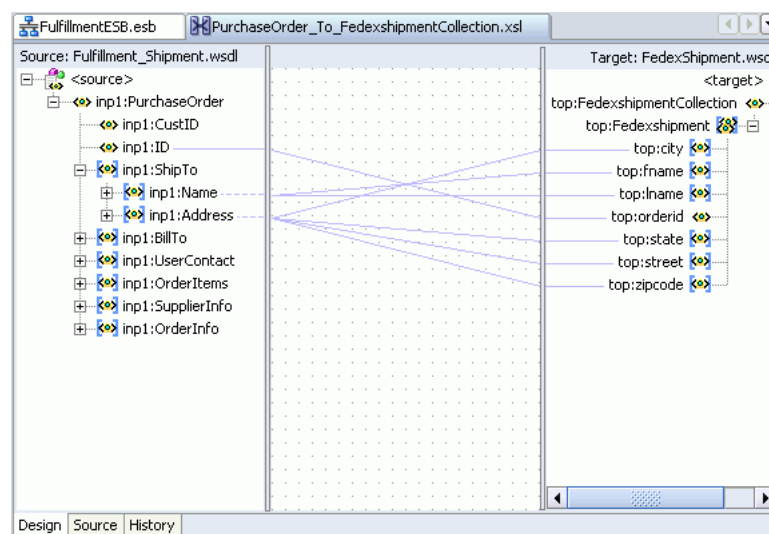
If you chose Create New Mapper File, the Data Transformation Mapper tool opens to enable you to correlate source schema elements to target schema elements.

5. You can select and drag a component on either side of the tool to the component you want to correlate on the other side of the mapper tool. When you initially select and drag, the Auto Map Preferences dialog displays so you can set preferences for the mapping. [Figure 6-10](#) is an example of an Auto Map Preferences dialog.

Figure 6–10 Auto Map Preferences Dialog

6. Use **File > Save** to save your updates.

Figure 6–11 is an example of a data transformation mapping in the Design tab of the Data Mapper tool. You can view the source code of the mapping in the Source tab.

Figure 6–11 Sample Data Mapping

For information about using the XSLT Data Mapper with Oracle Enterprise Service Bus, see "XSLT Data Mapper and Transformations" in *Oracle Enterprise Service Bus Developer's Guide*.

6.10.7 How to Add An Expression for Filtering Messages Based on Payload

In a routing rule, you can specify an expression to filter messages based on their payload. You can, for example, route messages for a customer record to different offices, based on that customer's postal code.

To specify a filter expression:

1. If you have not already done so, expand the Routing Rules information, as described in [Section 6.10.1, "How to Access Routing Rules"](#).
2. Click the **Add Filter Expression** icon, as shown in [Figure 6–8](#).
The Expression Builder opens.
3. Specify the filter expression, and then click **OK**.

The Expression Builder dialog displays.

You build the expression for filtering as follows:

1. In the WSDL Message box, expand the message definition and select the message element on which you want to base the expression.

Notice that the Content Preview box indicates the XPath expression for the selected WSDL message element.

2. Click **Insert Into Expression**.

The expression is presented in the Expression box.

3. From the Function box, select the function that you want to apply to the WSDL Message payload.

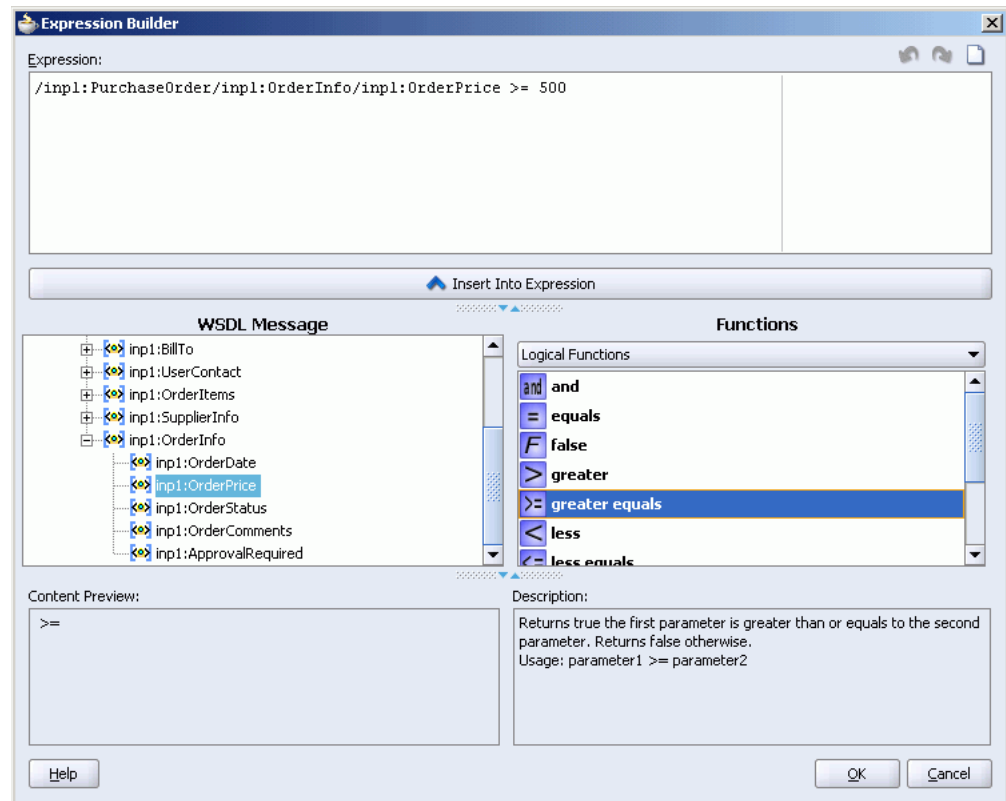
Functions are listed within categories that are listed when you click the down arrow within the Functions box. For example, if you click the down arrow and select Logical Functions, the list appears. Notice that if you select a function within the Logical Functions list, a description of that function is presented in the Explanation box.

4. Click **Insert Into Expression**.

The XPath expression for the selected function is inserted in to the Expression box. Notice that because the expression requires editing by hand to complete the expression, a red squiggle appears at the end of the line.

5. Complete the expression.

[Figure 6–12](#) is an example of a completed Expression Builder window.

Figure 6–12 Sample Expression Builder Tool

6. Validate the expression by clicking the green check mark.
7. If the expression is invalid or you need to make a change, you can edit the expression manually, or use the expression editing icons.
8. Click **OK**.

The expression is added to the Routing Rule property sheet.

To modify or delete a filter expression, double-click the Add Filter Expression icon, and then modify or delete the expression in the Expression panel of the Expression Builder.

6.10.8 What Happens When You Specify Routing Rules

After a routing service has been updated with specified routing rules, the routing service name with the extension `esbsvc` is updated in the Resources folder under the ESB project in the Application Navigator.

6.10.9 What Happens at Runtime

A message instance is processed according to the routing rules specified for the routing service. The rules determine the specific target for the message instance.

6.11 Adding a Domain-Value Map to Oracle Enterprise Service Bus

The Maps view of Oracle ESB Control enables you to create, update, and delete domain-value maps, as well as view, export, and import existing domain-value maps. For information on Oracle ESB Control, see [Section 12.1, "Introduction to Monitoring with Oracle ESB Control"](#).

Applications that you want to integrate using Oracle Enterprise Service Bus likely use different values to represent the same information. For example, one application might represent a state with the long name (Massachusetts) while another application may represent the state with an abbreviation (MA). A **domain-value map** enables you to associate values from one application with values from another.

Each domain-value map typically holds a specific category of value mappings among multiple applications. For example, one domain-value map might hold mappings for state codes and another might hold mappings for units of measurement.

After you create a domain-value map in Oracle ESB Control, a developer can use it when defining an data transformation map XSL file with the Oracle JDeveloper ESB Designer data mapper, so that application-specific values can be used at runtime. For example, suppose you want to use a domain-value map to perform a runtime lookup to convert long state names input to the two-letter state code output. In this scenario, the state name is passed to an Oracle Enterprise Service Bus flow. Within Oracle Enterprise Service Bus, the data is transformed by a transformation specified in the routing rule of the routing service from the state name to the state code using a domain-value map look up.

For more information, including exporting, importing and modifying domain-value maps, see "Domain-Value Maps" in *Oracle Enterprise Service Bus Developer's Guide*.

6.11.1 How to Create a New Domain-Value Map

You can create a domain-value map in the Map view of Oracle ESB Control.

To create a new domain-value map:

1. At the top of Oracle ESB Control, click **Maps**.
2. Click the Create down-arrow on the left side of the console.

Two choices are presented: **Create a new map** and **Import a new map**.

3. Click **Create a new map**.

The console refreshes and appears similar to the Map view in [Figure 6–13](#).

Figure 6–13 Oracle ESB Control Map View – Create New Map

ORACLE® ESB Control

About | Logout
Logged in as : oc4jadmin

Domain Value Maps

Create Delete

Name

New DVM

New DVM
Double-click name to edit

Export Import OK Reset

Type Point-to-Point (Unseeded)

Description DVM description

Mappings
Use 'Add' menu to add rows or columns. Double-click a column header to change column names. Double-click a row to edit it.

Add Delete

	Domain1	Domain2
1	value1	value2

4. Replace the default **New DVM** with a name for the domain-value map file. To edit **New DVM**, click in the name field in the right pane of the screen above the **Mappings** area.

After the domain-value map file has been saved, the map name cannot be changed.

5. In the **Description** field, enter a description of the domain-value map.
6. If you need additional columns, click the **Add down-arrow**, and then click **Column** once for each additional column you want to add. If you add too many columns, select the column and click **Delete**.

Each column represents a domain. If you will be using the domain-value map to map values among four domains, for example, you click **Column** twice to add two more columns. There are no restrictions on what can be considered a domain; you specify domains based on your needs. A domain might be, for example, a trading partner, an organization, a department, an application, and so on.

For example, add one column **Add > Column**.

7. Double-click a column name to change it from the default value of **Applicationn** to a more meaningful name. Each column name must be unique within the domain-value map.
8. If you need additional rows, click the **Add down-arrow**, and then click **Row** once for each additional row you want to add. If you add too many rows, select the row and click **Delete**, or leave it as-is. Empty rows are deleted when you save the domain-value map.
9. Double-click a row and enter values for the domain value map.
10. Repeat the previous step until you have entered the desired rows. You do not have to enter all rows that will be required by the applications. You can edit the domain-value map to add more rows later.

Figure 6–14, shows an example of a domain-value map that has been created.

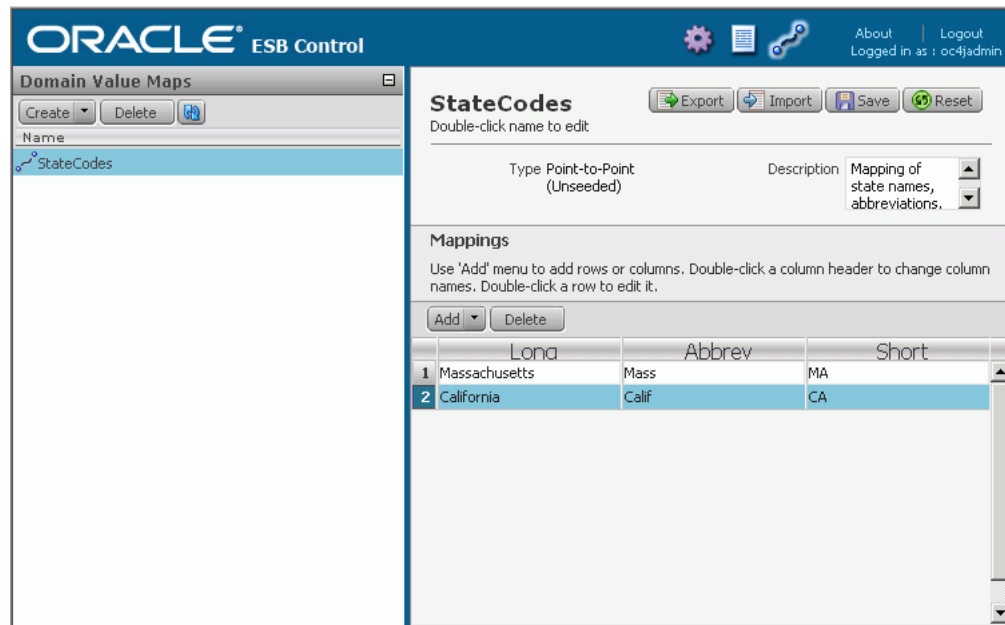
11. Review the name for the domain-value map and click the **Save** button above the Description field.

After you click the **Save** button, you cannot change the name for the domain-value map. However, you can change other components of the domain-value map.

Although these instructions indicate that you click **Save** when you have completed a domain-value map, you can click **Save** at any point in the process after specifying the name.

The completed domain-value map can now be used when creating a transformation in Oracle JDeveloper ESB Designer, as described in [Section 6.11.4, "How to Use a Domain-Value Map in a Data Transformation Map"](#).

Figure 6–14 Oracle ESB Control - Domain-Value Map Created



6.11.2 How to Export a Domain-Value Map

After you have created and saved a domain-value map, you can export it to a file. You might want to do this if you have created it on a test system and now want to export it so that you can import it on a production system.

To export a domain-value map:

1. In the Maps view of Oracle ESB Control, select the domain-value map you want to export in the left pane of the screen.
2. In right pane of the Maps view, click **Export**.
3. Click **OK**, and then wait for the operating system dialog box to open that enables you to save the file to the local file system. If you are prompted whether to open or save the file, select **Save**.
4. In the save file dialog box, selection the location where you want to save the domain-value map file, enter a name for the domain-value map file, and click **Save**.

5. In the Download complete dialog, click **Close**.

After the file has been exported and saved to the file system, you can edit the file with a text editor. However, make sure that you have carefully updated the file so that the file remains valid. After you have edited the file, you can import the updated domain-value map into Oracle ESB Control.

6.11.3 How to Import a Domain-Value Map

If you have an existing domain-value map on the local file system, you can import it into Oracle ESB Control as a means to add it to the ESB repository so that it can be used when you are specifying a transformation in a routing rule.

To import a domain-value map:

1. In the Maps view of Oracle ESB Control, click the **Create** down-arrow on the left side of the window.

Two choices are presented: **Create a new map** and **Import a new map**.

2. Click **Import a new map** to open the Import a New Map dialog.
3. In the Import a New Map dialog box, follow these steps:
 - a. In the Import field, enter the complete specification for the file on the local file system that you want to import or use Browse to locate the file.
 - b. Select or deselect **overwrite if a map with the same name already exists**, as desired.

Because the name becomes the primary key for the domain-value map when it is stored in the ESB repository, the name must be unique. If you select **overwrite if a map with the same name already exists** and a domain-value map with the same name already exists that you do not want to overwrite, you can do either of the following:

- Cancel this dialog box, rename the file on disk, and then restart the import operation.
- Deselect **overwrite if a map with the same name already exists**.

- c. Click **OK**.
4. Click **OK** in the Import Succeeded dialog.

The Console refreshes and displays the imported domain-value map file in the left pane.

The imported file is saved to the ESB repository and the domain-value map can now be used when creating a transformation in Oracle JDeveloper ESB Designer.

6.11.4 How to Use a Domain-Value Map in a Data Transformation Map

To include a domain-value map in an XSL data transformation, you drag and drop a `lookup-dvm` function from the component palette onto the connection between the source and target elements where you want the domain-value map to be used. For information on creating a data transformation maps, see [Section 6.10.6, "How to Create an XSL File for Data Structure Transformation"](#).

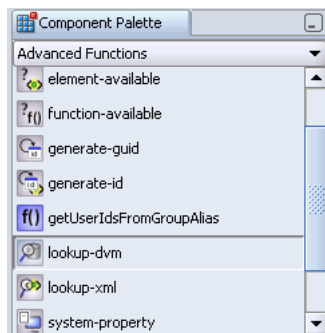
The following is an example of how to use a domain-value map in an XSL data transformation.

To use a domain-value map in a data transformation map:

1. In Oracle JDeveloper ESB Designer, open an ESB project.
2. In the Design tab, select a routing icon and double-click on the transformation XSL icon to display the Data Mapper window.
3. Expand the trees in the Source and Target panes in the Data Mapper window.
4. Drag and drop the source element to the target element where you want the domain-value map to be used.
5. In the Components palette, click the **down-arrow** and then select **Advanced Functions**.

The `lookup-dvm` function is listed in the component palette, as shown in [Figure 6-15](#).

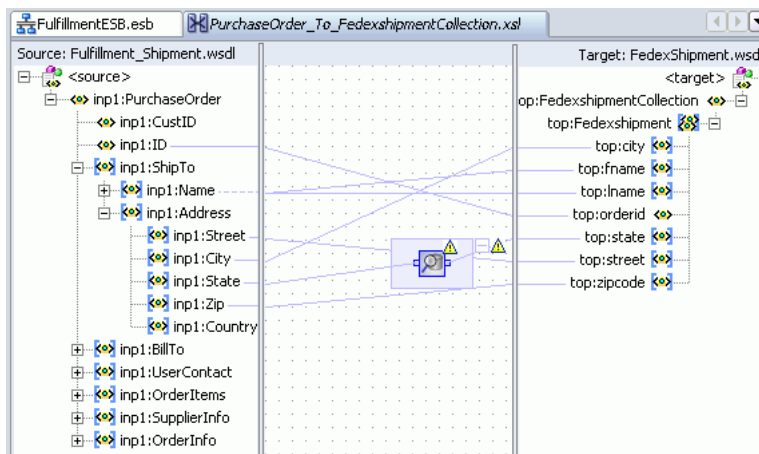
Figure 6-15 Component Palette – `lookup-dvm` Function



6. Drag and drop `lookup-dvm` onto the line that connects the source object to the target object.

A `lookup-dvm` icon appears on the connecting line, as shown in [Figure 6-16](#). The yellow warning triangles indicate that the definition of the `lookup-dvm` function is not complete.

Figure 6-16 Data Mapper – `look-up dvm` Function Added



7. Double-click the **look-up dvm** icon.

The Edit Function – `look-up dvm` dialog opens.

8. Specify values for the fields in the Edit Function – look-up dvm dialog, as shown in Figure 6–17.

Figure 6–17 Edit Function – lookup-dvm

Edit Function - lookup-dvm

Function Parameters:

dvmName: 'StateCodes'

sourceColumnName: 'Long'

sourceValue: /inpl:PurchaseOrder/inpl:ShipTo/inpl:Address/inpl:State

targetColumnName: 'Short'

defaultValue: ***

String Literals should be enclosed within ', (Example: \'abc\' or \\'abc\').

Function Description:

Returns a string by looking up the value for target column in the DVM where the value for source column is equal to the source value. The source value is an xpath expression bound to the source document of the XSLT transformation. The expression will be evaluated during the transformation and the result value will be passed as the source value for lookup.

Usage: orcl:lookup-dvm(dvmName as string, sourceColumn as string, targetColumn as string)

Buttons: Add, Remove, Move Up, Move Down, Help, OK, Cancel

- a. You can manually enter a value in **sourceValue** or the following methods:
 - Move the source value that appears in the **dvmName** field to the **sourceValue** field by clicking **Move Down** twice.
 - Press **Ctrl-Space** to launch Expression Builder. Press the up and down keys to locate an object in the list and press enter to select that item.
 - b. In the **dvmName** field, enter the name of the domain-value map schema that you previously defined using Oracle ESB Control.
 - Click the flashlight icon to the right of the dvmName field to select the name from a list of domain-value maps stored in the ESB repository.
 - c. In the **sourceColumnName** field, enter the name of the column in the domain-value map that is associated with the source element value.
 - Click the flashlight icon to the right of the sourceColumnName field to select the name from the columns defined for the domain-value map you previously selected.
 - d. In the **targetColumnName** field, enter the name of the column in the domain-value map that is associated with the target element value.
 - Click the flashlight icon to the right of the targetColumnName field to select the name from the columns defined for the domain-value map you previously selected.
 - e. Enter an optional default value.
 - f. Click **OK**.
9. From the **File** menu, click **Save All**.
 10. Register the project with the integration server or ESB Server.

When the transformation is included in an ESB routing rule, the transformation is applied at runtime.

You can view the results of a transformation if you have set up the field in the Trackable Fields tab of the Oracle ESB Control Services view. The tracking data is displayed in the Diagram tab of the Oracle ESB Control Instances view. See [Section 12.9, "Defining Trackable Fields"](#).

6.12 Registering ESB Projects and Services with the ESB Server

After you have completed the design of an ESB project in Oracle JDeveloper ESB Designer, you register the project and its services with the ESB server.

The process of registration is required to run the services and to view and manage these services in Oracle ESB Control, which is used to monitor ESB during runtime. For information about Oracle ESB Control, see [Chapter 12, "Monitoring Oracle Enterprise Service Bus"](#).

When you make any updates to the ESB project and its services, you need to register the project again with the integration server to ensure that the changes are implemented during runtime.

6.12.1 How to Register ESB Services with the ESB Server

You use the Register with ESB option in Oracle JDeveloper ESB Designer with an integration server connection to complete the registration. Before registering ESB projects and services, you need to have an integration server connection set up between Oracle JDeveloper ESB Designer and the server. For information on creating connections, see [Section 2.2.7, "Deploying the Separate SOA Order Booking Applications"](#).

Note: Before registering an ESB project with the ESB Server or integration server, you must have the server must be running.

To register ESB services with the ESB Server:

1. In the Application Navigator, right-click on the ESB project and choose **Register with ESB**.
2. Click on the local integration server connection or a connection to another integration server that has been set up.
3. When the ESB Registration Summary dialog displays, click **OK**.

After a successful registration, the information for the services is written into the ESB Server repository. When you refresh Oracle ESB Control, the information about the ESB services that you registered is retrieved and the services are displayed in the Service Navigation tree of the console. See [Section 6.3.2, "What Happens When You Create an ESB Project"](#).

For information about Oracle ESB Control, see [Chapter 12, "Monitoring Oracle Enterprise Service Bus"](#).

6.13 Testing and Validating the Oracle Enterprise Service Bus Flow

To test and validate the Oracle Enterprise Service Bus flow, you can use Oracle Enterprise Manager or other testing tools to invoke any of the ESB services over SOAP. You can test an individual outbound service or an entire flow by invoking a routing service that invokes multiple outbound services.

Note: ESB provides runtime error handling capabilities for error logging, notifications on errors, and automatic and manual retries on errors. For information on runtime error handling with the Oracle ESB Control, see [Section 12.11, "Error Handling in Message Instance Processing"](#).

6.13.1 How to Test ESB Services Using Oracle Enterprise Manager

You can use Oracle Enterprise Manager Application Server Control to test ESB services while you are designing the services in Oracle JDeveloper ESB Designer. Before testing a service in an ESB project, register the project with the integration server or local ESB server. See [Section 6.12.1, "How to Register ESB Services with the ESB Server"](#).

To test the services in the ESB project you are creating in Oracle JDeveloper ESB Designer, use the Test Web Service feature of Oracle Enterprise Manager.

To test using Oracle Enterprise Manager:

1. Open Oracle Enterprise Manager Application Server Control for your system.
2. Enter the username and password to log in.
3. Click on an application server in the list of All Application Servers.
For example, click on the **home** application server.
4. In Application Server page, click on the **Web Services** link.

The list of Web services displays, as shown in [Figure 6–18](#).

Figure 6–18 Oracle Enterprise Manager Web Services

ORACLE Enterprise Manager 10g
Application Server Control

Cluster Topology > Application Server: soademo.erbelden-pc2.us.oracle.com > OC4J: home > Application: esb-rt

Page Refreshed Jun 27, 2006 11:10:31 AM PDT

Home Web Services Performance Administration

Test Service

Select	Port Name	Web Service	Application Status	Port Enabled	Start Time	Invocations
☐	soap_FedexShippment_FedexShippment_ptt	FedexShippment_pttService	↑	✓	Jun 27, 2006 9:34:44 AM PDT	
☐	soap_FulfillmentBatch_sendMessage_ptt	sendMessage_pttService	↑	✓	Jun 27, 2006 9:34:45 AM PDT	
☒	soap_OrderFulfillment_execute_ppt	execute_pptService	↑	✓	Jun 27, 2006 9:34:46 AM PDT	

5. Select a Web service and click the **Test Service** button.

The Test Web Service page displays, as shown in Figure 6–19.

Figure 6–19 Oracle Enterprise Manager Test Web Service

ORACLE Enterprise Manager 10g
Application Server Control

Cluster Topology > Application Server: soademo.erbelden-pc2.us.oracle.com > OC4J: home > Application: esb-rt > Web Service: soap_OrderFulfillment_execute_ppt > Test Web Service: execute_pptService

Page Refreshed Jun 27, 2006 10:34:05 AM PDT

Discovered Websites

This table shows websites that may be used to test this Web Service.

Select	Listener	Protocol	Host	Port
☒	J2EE Website	http	erbelden-pc2	8888

URL: Test Web Service

✓ **TIP** The URL field may be modified to test the web service using a custom URL.

Setup | Logs | Help | Logout

Copyright © 1996, 2006, Oracle. All rights reserved.
Oracle, JD Edwards, PeopleSoft, and Retek are registered trademarks of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.
[About Oracle Enterprise Manager 10g Application Server Control](#)

6. In the Test Web Service page, click the Test Web Service button.

The test page displays.

7. Enter data in the fields of the Test page to test the service, as shown in Figure 6–20.

Figure 6–20 Oracle Enterprise Manager Test Page**execute_pptService endpoint**

For a formal definition, please review the [Service Description](#).

Download the JavaScript Stub (BETA) for [soap_OrderFulfillment_execute_ppt](#) and see its [documentation](#).

soap_OrderFulfillment_execute_ppt

Operation : ☒ HTML Form ☐ XML Source

☒ Reliable Messaging ☐ Include In Header

☒ WS-Security ☐ Include In Header

☒ PurchaseOrder

CustID xsd:string

ID xsd:string

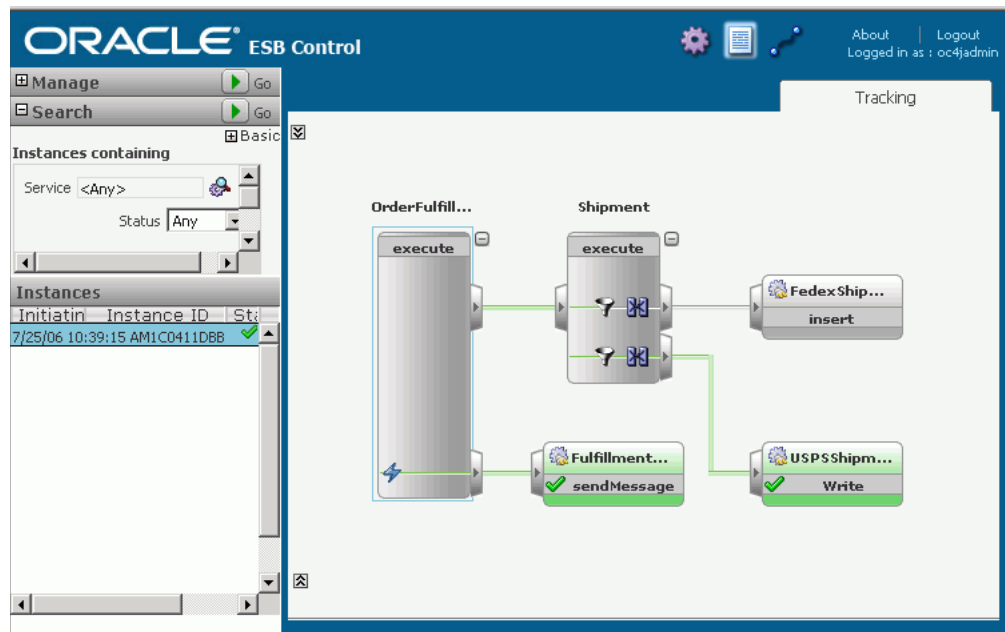
☒ ShipTo ☒ Include in Message

☒ Name ☒ Include in Message

First xsd:string ☒ Include in Message

Last xsd:string ☒ Include in Message

8. After providing sample data, click **Invoke** to display the Test Results page.
Check whether there are any errors in the page. You can close the test page after viewing the results.
9. View Instances in the Oracle ESB Control to check whether the message instance was successfully processed with the sample data, as shown in [Figure 6–21](#). For information on the Instances view of the Oracle ESB Control, see [Section 12.6, "Using the Instances View to Monitor Message Instances"](#).
10. If necessary, correct any errors and repeat the test process.

Figure 6–21 Oracle ESB Control - Instances View with Successful Processing

6.13.2 What Happens When You Test Using Oracle Enterprise Manager

When you test with Oracle Enterprise Manager, you invoke the Web service associated with an ESB project service using the sample values for the message instance. This approximates the runtime processing of the message instance.

You can monitor whether the message instance was successfully processed in the Instances View of the Oracle ESB Control. See [Section 12.6, "Using the Instances View to Monitor Message Instances"](#).

Using Oracle BPEL Process Manager

This chapter describes how to use Oracle BPEL Process Manager to create business process workflows.

This chapter includes the following sections:

- [Section 7.1, "Introduction to Oracle BPEL Process Manager"](#)
- [Section 7.2, "Creating an Application and a BPEL Project"](#)
- [Section 7.3, "Invoking a Web Service"](#)
- [Section 7.4, "Using an Assign Activity"](#)
- [Section 7.5, "Creating a Transformation"](#)
- [Section 7.6, "Communicating with a Database"](#)
- [Section 7.7, "Creating Conditional Branching Using a Switch Activity"](#)
- [Section 7.8, "Using Business Rules and a Decide Activity"](#)
- [Section 7.9, "Creating a Parallel Flow"](#)
- [Section 7.10, "Handling Faults"](#)
- [Section 7.11, "Using Sensors for Monitoring"](#)
- [Section 7.12, "Creating a Human Workflow Task"](#)
- [Section 7.13, "Creating an Email Notification"](#)
- [Section 7.14, "Compiling and Deploying a BPEL Process"](#)

7.1 Introduction to Oracle BPEL Process Manager

Oracle BPEL Process Manager provides a framework for designing, deploying, and managing cross-application business processes based on business process execution language (BPEL) standards. BPEL is an XML-based language for creating business process workflows and orchestrating XML-based web services. The resulting BPEL application supports a service-oriented architecture (SOA) paradigm by leveraging web computing standards to deliver an application that *is* the business process.

In a BPEL process, every piece of data is in XML format. This includes the messages passed to and from the BPEL process, the messages exchanged with external services, and all variables. You define the types for these messages and variables with the XML schema, in the web services description language (WSDL) file for the BPEL process itself or in the WSDL files for the web services that a BPEL process invokes. The WSDL defines the input and output messages for a BPEL process, the supported client interface and operations, and other features. This functionality enables a BPEL process to be called as a web service. XML variables are manipulated in the BPEL code, including data transformation between different formats required by web services and local manipulation of data (for example, to combine the results from several web service invocations).

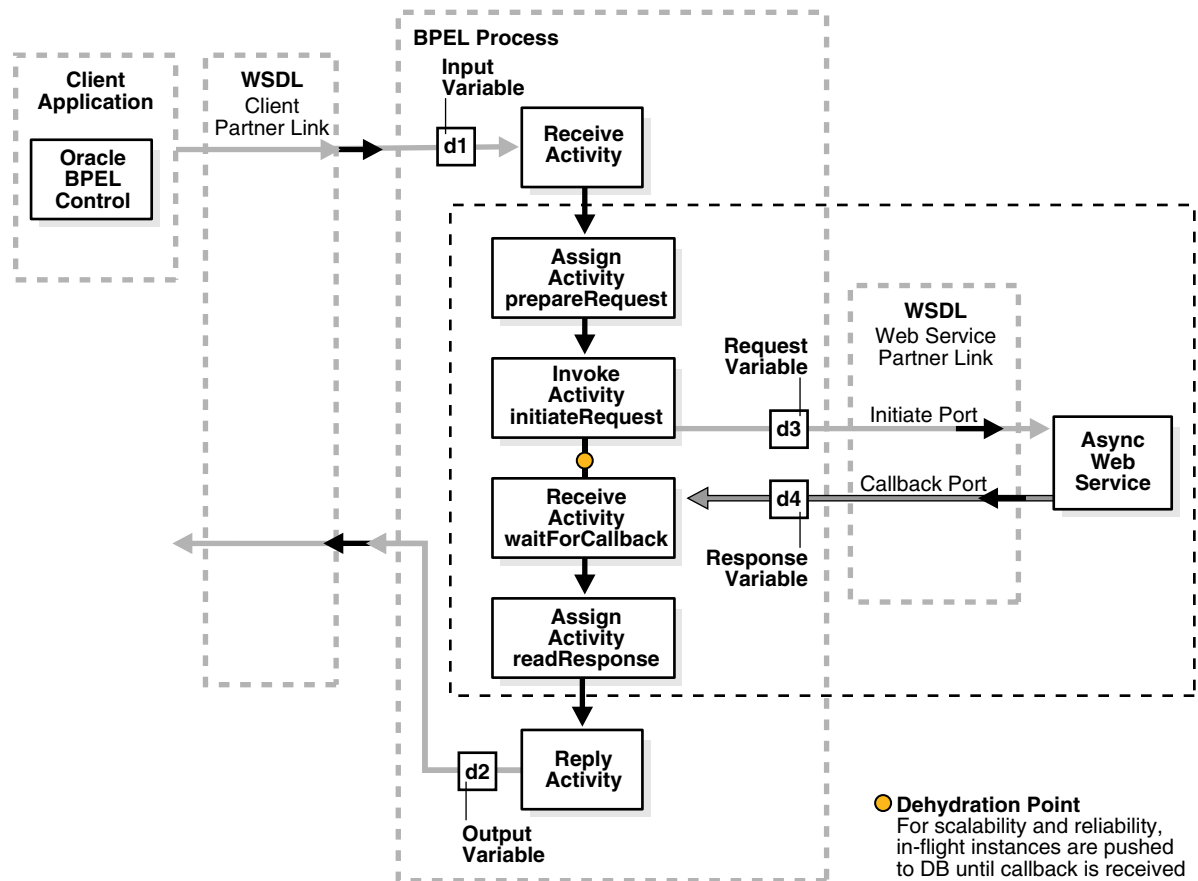
When designing a BPEL process, you model the business process by specifying the BPEL building blocks, *activities* and *services*, in the order required to execute the business workflow. This chapter describes the key activities and services that you need to understand the BPEL components of the SOA Order Booking application, as well as basic topics such as how to create a BPEL project and invoke a web service. See the following for the context for understanding the activities and services described in this chapter:

- [Section 2.3, "An Overview of the SOA Order Booking Application"](#) on page 2-17
- [Figure 2-26, "The SOAOrderBooking BPEL Flow"](#) on page 2-33

See *Oracle BPEL Process Manager Developer's Guide* for a more complete description of all the BPEL activities and services.

[Figure 7-1](#) shows how a BPEL process receives a request from a client application, invokes a web service to process that request, and returns a value to the client. The figure depicts an asynchronous web service invocation, which returns a result to the client using a callback. Asynchronous processes are good for long-running production environments and do not require the client or partner link to wait for a response.

Figure 7-1 BPEL Process and Asynchronous Service Invocation



Within the BPEL process, between the receive and reply activities, the BPEL process *assigns* the client request to an input variable (`prepareRequest`), *invokes* the request by putting the contents of this request into a request variable and sending the request to the web service (`initiateRequest`), *receives* the response back from the web service (`waitForCallback`), and *assigns* the response to an output variable to read the reply (`readResponse`). Oracle BPEL Process Manager automatically performs dehydration (stores the process in the dehydration store database) after an invoke activity in an asynchronous process, as shown in Figure 7-1. See *Oracle BPEL Process Manager Installation Guide* for information about setting up a dehydration database.

BPEL processes include partner links to various web services. Partner links define the external services with which the BPEL process interacts. A partner link type characterizes the conversational relationship between two services by defining the roles played by each service in the conversation and specifying the port type provided by each service to receive messages within the context of the conversation. In the SOA Order Booking application, the client partner link kicks off the BPEL process with order information that is routed by Oracle Enterprise Service Bus. See [Chapter 2, "The SOA Order Booking Application"](#) for a description of the BPEL processes in the SOA Order Booking application.

See the following for more information about Oracle BPEL Process Manager:

- *Oracle BPEL Process Manager Administrator's Guide*
- *Oracle BPEL Process Manager Developer's Guide*
- *Oracle BPEL Process Manager Installation Guide*
- *Oracle BPEL Process Manager Order Booking Tutorial*
- *Oracle BPEL Process Manager Quick Start Guide*
- *Oracle BPEL Process Manager Client API Reference*
- *Oracle BPEL Process Manager Sensor API Reference*
- *Oracle BPEL Process Manager Workflow Services API Reference*

Read this chapter to understand:

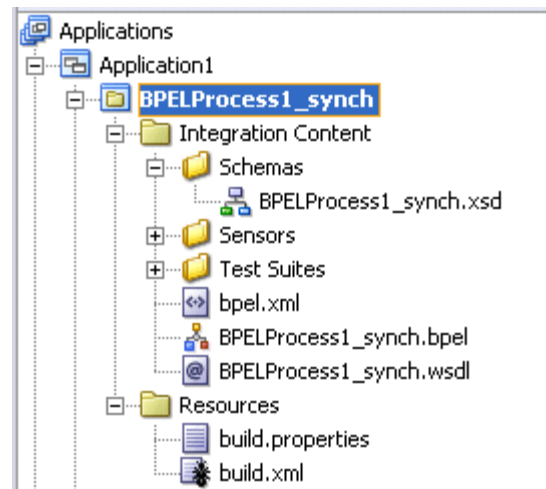
- How to create BPEL processes
- How to communicate with web services
- How to deploy and run BPEL processes

7.2 Creating an Application and a BPEL Project

When creating a BPEL process, the first step is to create an application and a BPEL project in Oracle JDeveloper. The following files are generated when you do so.

- *bpel_process_name.bpel*
A BPEL source file that contains the basic, required activities for a BPEL process. You build on this file when you drag and drop activities and services, create variables and partner links, and so on, in Oracle JDeveloper.
- *bpel_process_name.xsd*
An XML schema for validating the XML documents involved in the process. Custom XSD files that you create when you create a BPEL process or that you import appear in the same location as this file.
- *bpel_process_name.wsdl*
A basic WSDL for building the communication endpoints for a web service.
- *bpel.xml*
The deployment description file that defines the locations of the WSDL files for web services that are called by the BPEL process. The file references the public interface for the web service.

Figure 7–2 shows these files for a BPEL project named **BPELProcess1_synch**.

Figure 7-2 BPEL Project Files Created in Oracle JDeveloper

In addition to the files shown in [Figure 7-2](#), the following are also created:

- *application_name.jws*
The Java application file. For a complex business process, the application contains a number of projects (.jpr files).
- *bpel_process_name.jpr*
The Java project file. Each .jpr file is associated with a .bpel file with the same name.

7.2.1 How to Create an Application

Create a BPEL process by first creating an application in Oracle JDeveloper.

To create an application:

1. In the Application Navigator, right-click **Applications** and choose **New Application**.
2. Use the Create Application dialog to enter the name and location for the new application and to specify the application template.
 - a. Enter an application name.
 - b. Enter or browse for a directory name, or accept the default.
 - c. Enter an application package prefix.
The prefix, followed by a period, applies to objects created in the initial project of an application.
 - d. For a BPEL project, select **No Template [All Technologies]** for an application template.
 - e. Click **OK**.
3. In the Create Project window, click **Cancel**.

7.2.2 How to Create a BPEL Project

After creating an application in Oracle JDeveloper, you create one or more BPEL projects within the application.

To create a BPEL project:

1. Right-click the application name you just created and choose **New Project**.
2. Under **Categories**, expand **General** and select **Projects**.
3. Under **Items**, select **BPEL Process Project** and click **OK**.
4. Use the BPEL Project Creation Wizard - Project Settings dialog to set up the BPEL project.
 - a. Enter a name for the BPEL process.
 - b. Enter a namespace or accept the default.
 - c. Use the default project settings or specify your own.
 - d. Select a template for one of the following:

Asynchronous BPEL process: Creates an asynchronous process with a default receive activity to initiate the BPEL process flow and an invoke activity to asynchronously call back the client.

Synchronous BPEL process: Creates a synchronous process with a default receive activity to initiate the BPEL process flow and a reply activity to return the results.

Empty BPEL process: Creates an empty process with no activities.
 - e. Click **Next**.
5. Use the BPEL Project Creation Wizard - Input/Output Elements dialog to add your own schema files or accept the default schema files.

7.2.3 What Happens When You Create an Application and a BPEL Project

After you create a BPEL project, the skeleton BPEL process appears in the Oracle JDeveloper diagram view, as shown in [Figure 7-3](#). This is the workflow represented by the `bpel_process_name.bpel` file.

[Figure 7-3](#) shows a synchronous BPEL process. The receive activity, labeled **receiveInput**, is used for the input variable, and the reply activity, labeled **replyOutput**, is used for the output variable.

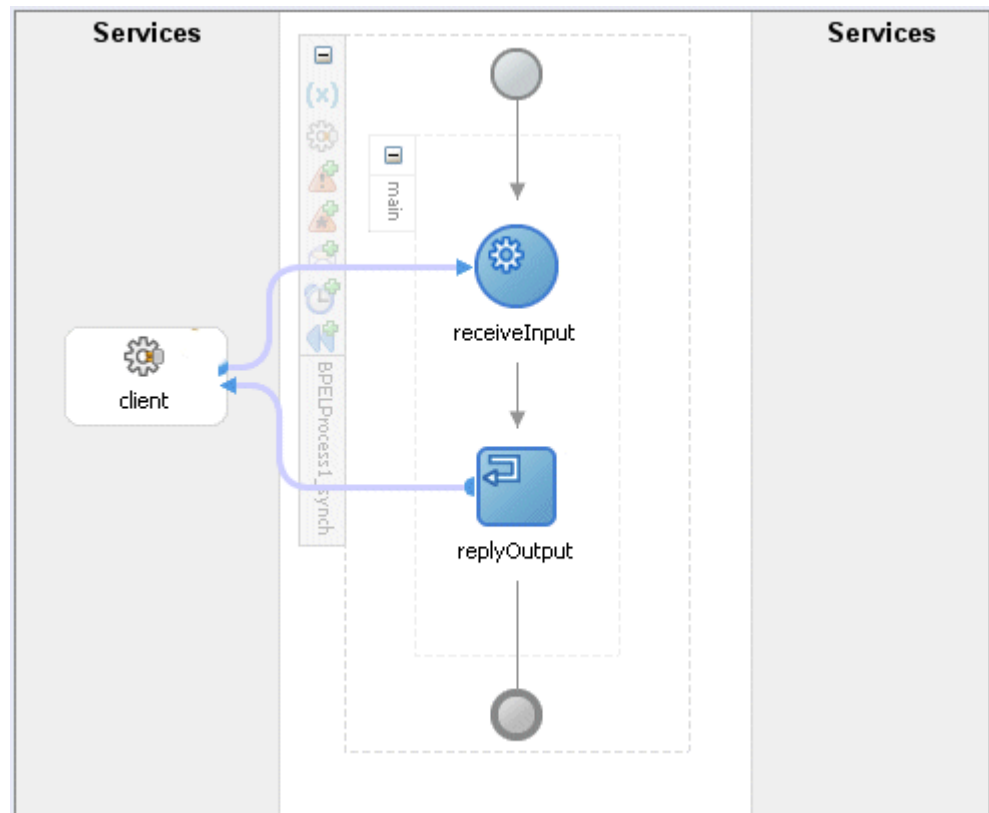
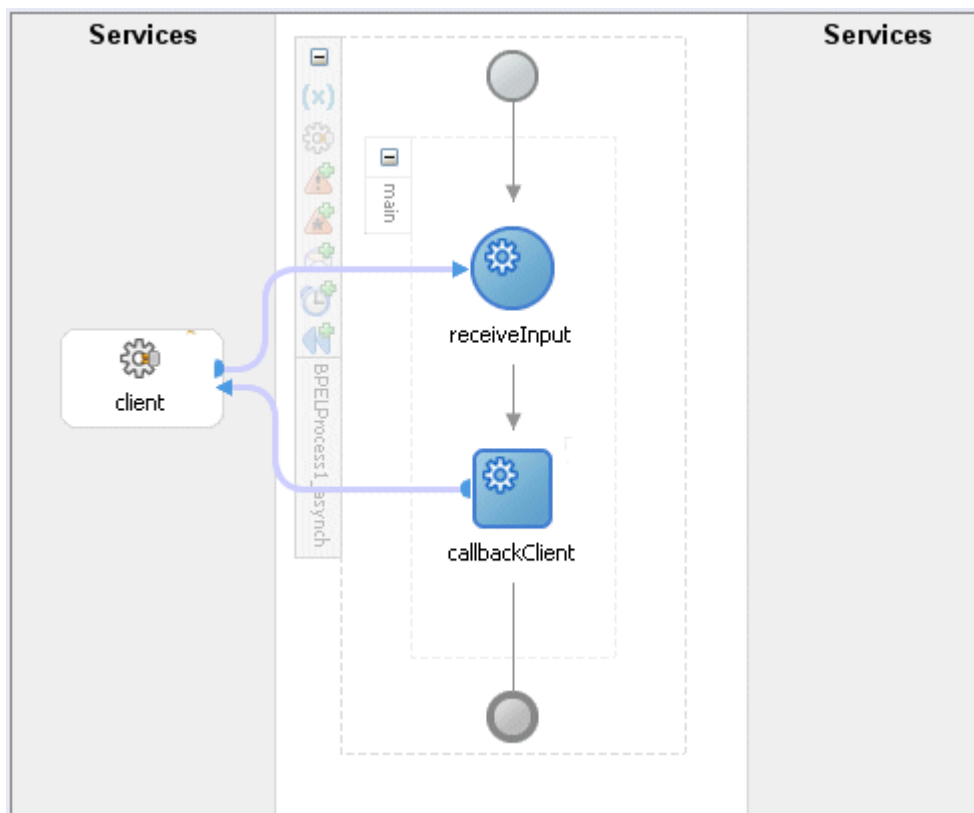
Figure 7–3 BPEL Diagram for a Synchronous Process

Figure 7–4 shows an asynchronous process. The receive activity, labeled **receiveInput**, is used for the input variable, and the invoke activity, labeled **callBackClient**, is used for the output variable and to invoke the web service.

Figure 7–4 BPEL Diagram for an Asynchronous Process

You can view the BPEL source code corresponding to the diagram view from the **Source** tab below the diagram.

[Example 7–1](#) shows the source code for an asynchronous process.

Example 7–1 Source Code for an Asynchronous Process

```
<?xml version = "1.0" encoding = "UTF-8" ?>
<!--
////////////////////////////////////////////////////////////////
Oracle JDeveloper BPEL Designer
Created: Wed May 03 10:14:38 PDT 2006
Author:
Purpose: Asynchronous BPEL Process
////////////////////////////////////////////////////////////////
-->
<process name="BPELProcess1_asynch"
  targetNamespace="http://xmlns.oracle.com/BPELProcess1_asynch"
  xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
  xmlns:xp20="http://www.oracle.com/XSL/Transform/java/
    oracle.tip.pc.services.functions.Xpath20"
  xmlns:bpws="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
  xmlns:ldap="http://schemas.oracle.com/xpath/extension/ldap"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:client="http://xmlns.oracle.com/BPELProcess1_asynch"
  xmlns:bpelx="http://schemas.oracle.com/bpel/extension"
  xmlns:ora="http://schemas.oracle.com/xpath/extension"
  xmlns:orcl="http://www.oracle.com/XSL/Transform/java/
    oracle.tip.pc.services.functions.ExtFunc">
```



```

<!--

////////////////////////////////////
PARTNERLINKS
List of services participating in this BPEL process
////////////////////////////////////
-->
<partnerLinks>
  <!--
  The 'client' role represents the requester of this service. It is
  used for callback. The location and correlation information associated
  with the client role are automatically set using WS-Addressing.
  -->
  <partnerLink name="client" partnerLinkType="client:BPELProcess1_asynch"
    myRole="BPELProcess1_asynchProvider"
    partnerRole="BPELProcess1_asynchRequester"/>
</partnerLinks>
<!--

////////////////////////////////////
VARIABLES
List of messages and XML documents used within this BPEL process
////////////////////////////////////
-->
<variables>
  <!-- Reference to the message passed as input during initiation -->
  <variable name="inputVariable"
    messageType="client:BPELProcess1_asynchRequestMessage"/>
  <!-- Reference to the message that will be sent back to the requester during callback -->
  <variable name="outputVariable"
    messageType="client:BPELProcess1_asynchResponseMessage"/>
</variables>
<!--

////////////////////////////////////
ORCHESTRATION LOGIC
Set of activities coordinating the flow of messages across the
services integrated within this business process
////////////////////////////////////
-->
<sequence name="main">
  <!-- Receive input from requestor. (Note: This maps to operation defined in BPELProcess1_
asynch.wsdl) -->
  <receive name="receiveInput" partnerLink="client"
    portType="client:BPELProcess1_asynch" operation="initiate"
    variable="inputVariable" createInstance="yes"/>
  <!--
  Asynchronous callback to the requester. (The callback location and correlation id are
  transparently handled using WS-addressing.)
  -->
  <invoke name="callbackClient" partnerLink="client"
    portType="client:BPELProcess1_asynchCallback"
    operation="onResult" inputVariable="outputVariable"/>
</sequence>
</process>

```

See *Oracle BPEL Process Manager Quick Start Guide* for more information about the source code for an asynchronous process.

7.3 Invoking a Web Service

A web service can be invoked as a synchronous or asynchronous operation. Synchronous web services provide an immediate response to a query, and block the BPEL process for the duration of the operation. Asynchronous web services do not block the BPEL process, and are useful for environments in which a service can take a long time to process a client request. Asynchronous services also provide a more fault-tolerant and scalable architecture than synchronous services.

Both synchronous and asynchronous web services require the following:

- **Partner link:** Defines the location and the role of the web services with which the BPEL process interacts, as well as the variables used to carry information between the web service and the BPEL process. A partner link is required for each web service that the BPEL process calls.
- **Invoke activity:** Opens a port in the BPEL process to send and receive data. A BPEL process uses this port to send the required data and return a response. For a synchronous web service, only one port is needed for both the send and receive functions. Because the BPEL process waits for the reply, nothing that explicitly receives the reply is needed.

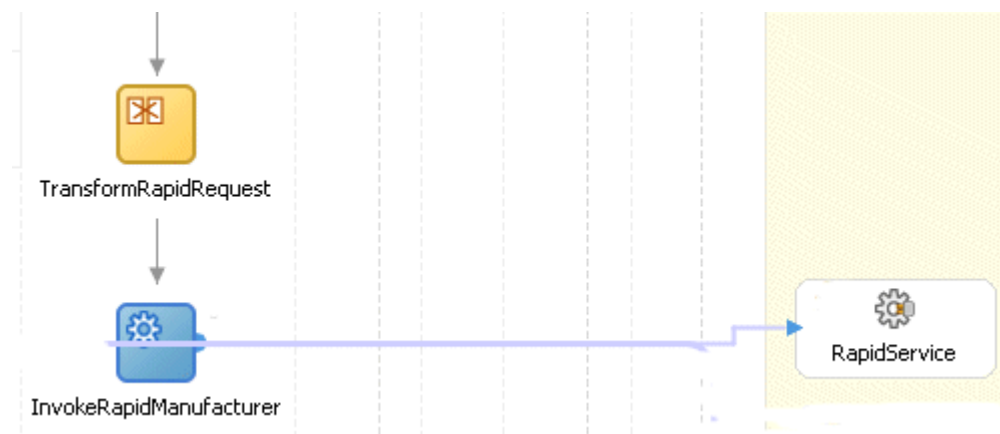
An asynchronous web service also requires the following:

- **Receive activity:** Provides another port to receive the reply from the web service.
- **Correlation ID:** Ensures that the response returned corresponds to the request. The correlation ID is unique to the client and partner link initiating the request and is sent to the web service.

The BPEL process in the SOA Order Booking application includes a synchronous web service invocation (to Rapid Manufacturer) and an asynchronous web service invocation (to Select Manufacturer).

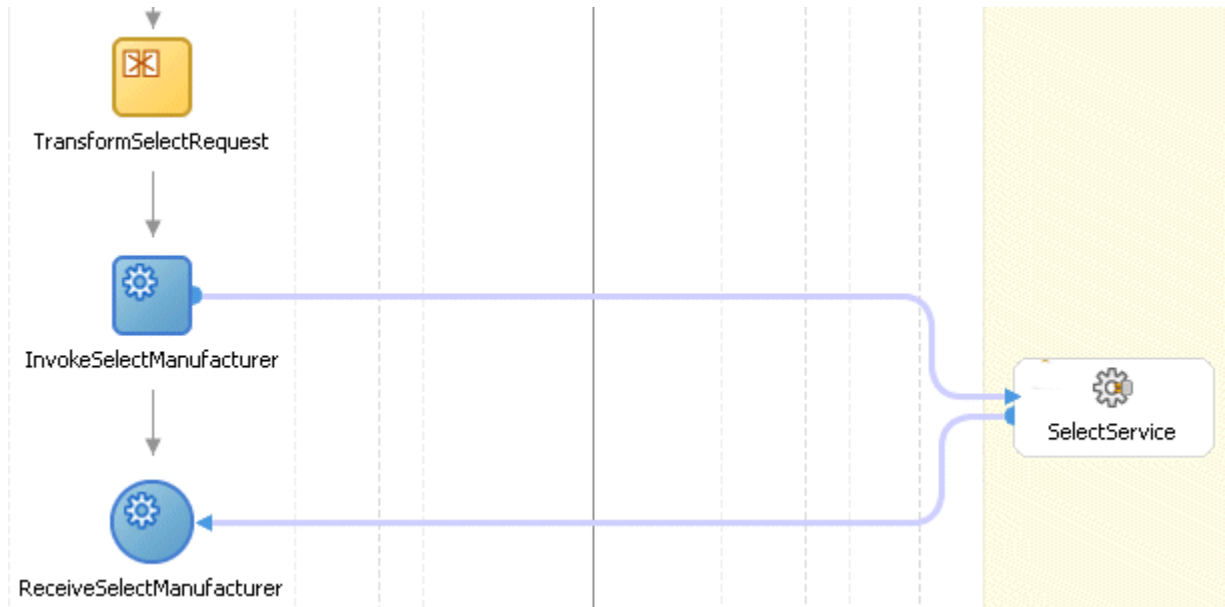
A synchronous invocation is shown in [Figure 7-5](#). The invoke activity (**InvokeRapidManufacturer**) solicits a quote using the **RapidService** partner link. The Rapid Manufacturer web service (not shown in the figure) replies immediately to the query.

Figure 7-5 Synchronous Invocation to Rapid Manufacturer



An asynchronous invocation is shown in [Figure 7-6](#). The invoke activity (**InvokeSelectManufacturer**) solicits a quote using the **SelectService** partner link. The Select Manufacturer web service (not shown in the figure) processes the request and returns a reply, which is tracked by the correlation ID, to the **ReceiveSelectManufacturer** activity.

Figure 7-6 Asynchronous Invocation to Select Manufacturer



7.3.1 How to Invoke a Web Service

For both synchronous and asynchronous web service invocations, you add the following activities to a BPEL process:

- A partner link activity to define the external web service with which the BPEL process interacts
- A scope activity for nesting related activities that can have their own local variables and fault handlers
- An invoke activity to specify the operation you want to invoke for the web service (identified by its partner link)
- Two assign activities. The first assign activity takes the client query input and contacts the web service. The second assign activity returns the response from the web service.

For an asynchronous web service invocation, you also add the following:

- A receive activity after the invoke activity, as shown in [Figure 7-6](#). The receive activity receives the data from the web service. You use the Receive dialog to select the partner link, operation, and variable that are provided by the web service WSDL.

7.3.1.1 Creating a Partner Link

A partner link defines the external services with which the BPEL process interacts.

To create a partner link to a web service:

1. Create an asynchronous BPEL process.
See [Section 7.2.1, "How to Create an Application"](#) on page 7-5 for instructions.
2. Drag and drop a **PartnerLink** activity under one of the **Services** swim lanes.
3. Use the **General** tab of the Create Partner Link dialog to enter values to create a partner link, as shown in [Figure 7-7](#).

Figure 7-7 Creating a Partner Link for an Asynchronous Web Service

The screenshot shows the 'Create Partner Link' dialog box with the 'General' tab selected. The 'Name' field is 'SelectService' and the 'Process' field is 'SOAOrderBooking'. Under the 'WSDL Settings' section, the 'WSDL File' is 'http://localhost:8888/orabpel/default/SelectManufact'. The 'Partner Link Type' is 'ns17:SelectService_PL', the 'Partner Role' is 'SelectServiceProvider', and the 'My Role' is 'SelectServiceRequester'.

- a. Enter a name for the partner link.
 - b. Use the **Service Explorer** (flashlight icon) to browse for the URL of the web service WSDL file.
This WSDL describes the web service communication endpoints.
 - c. Select a partner link type.
This value is supplied by the web service WSDL.
 - d. Select a partner role.
This is the *provider* role of the web service. This value is supplied by the web service WSDL.
 - e. For a synchronous web service invocation, skip the **My Role** field. A role is not required for a partner link to a synchronous web service.
For an asynchronous web service invocation, select a value for the **My Role** field on the Create Partner Link dialog, as shown in [Figure 7-7](#). This is the *requester* role of the BPEL process. This value is supplied by the web service WSDL and is needed for the Select Manufacturer web service to return a response to the BPEL process, **CallSelectManufacturer**.
 - f. Click **OK**.
4. Save the file.

7.3.1.2 Creating a Scope Activity

A scope activity is a container for a group of activities that you want to process as one unit.

To create a scope activity:

1. Drag and drop a **Scope** activity between the **receiveInput** and **callbackClient** activities.
2. Double-click the **Scope** activity.
3. Select the **General** tab and enter a name for the **Scope** activity.
4. Click **OK**.
5. Save the file.

7.3.1.3 Creating an Invoke Activity Inside a Scope Activity

An invoke activity sends information, typically a request, to a web service, which is identified by its partner link, and specifies an operation for the web service to perform.

To create an invoke activity inside a scope activity:

1. Expand the **Scope** activity that you just created.
2. Drag and drop an **Invoke** activity into the **Scope** activity.
3. Double-click the **Invoke** icon.
4. Use the Invoke dialog to identify the initial interaction operation between the BPEL process and the web service.
 - a. Enter a name for the **Invoke** activity.
 - b. Browse for the partner link that you previously created.
 - c. Select an operation.

This value is supplied by the web service WSDL.

- d. Create an input variable with the first icon (the automatic variable creation icon) to the right of the **Input Variable** field.

A variable named *invokeName_operationName_InputVariable* is automatically created in the **Input Variable** field. The variable is automatically assigned a message type of *WebServiceNameRequestMessage*. This value is supplied by the web service WSDL. This variable holds the contents of the request and is sent to the web service.

- e. Create an output variable with the first icon to the right of the **Output Variable** field.

A variable named *invokeName_operationName_OutputVariable* is automatically created in the **Output Variable** field. This variable is automatically assigned a message type of *WebServiceNameResponseMessage*. This value is supplied by the web service WSDL. This variable holds the contents of the reply from the web service.

5. Save the file.

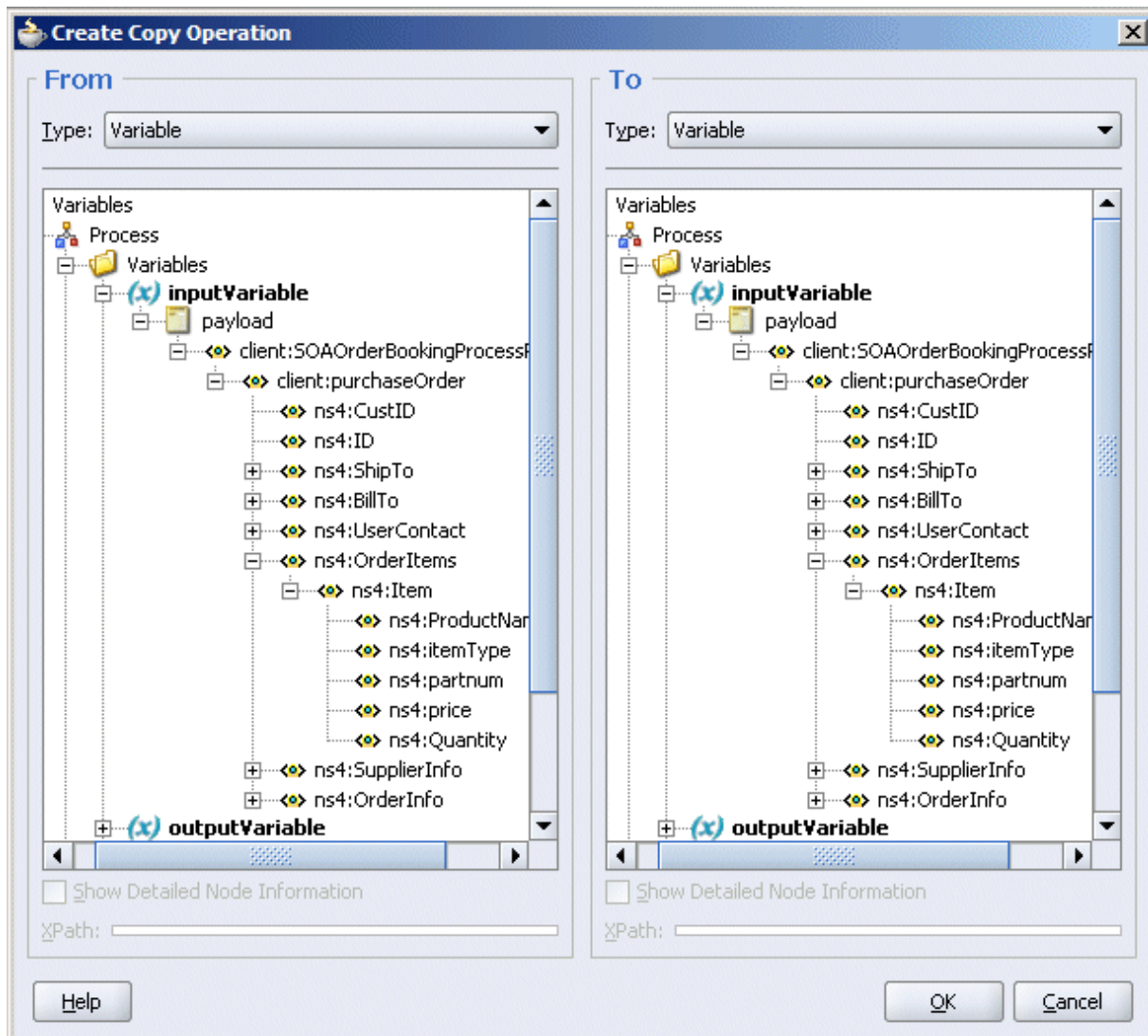
7.3.1.4 Creating an Initial Assign Activity Inside a Scope Activity

An assign activity transfers data between variables, expressions, and other elements.

To create an initial assign activity inside a scope activity:

1. Drag and drop an **Assign** activity to above the **Invoke** activity and inside the **Scope** activity.
2. Double-click the **Assign** icon.
3. Use the **General** tab of the Assign dialog to enter a name for the **Assign** activity.
4. Click **Apply**.
5. Click the **Copy Operation** tab.
6. From the **Create** list, select **Copy Operation**.
7. Use the Create Copy Operation dialog, shown in [Figure 7-8](#), to copy the contents of one variable to another. This **Assign** activity takes the client's purchase order as input and contacts the Rapid Manufacturer web service for a price quote.

Figure 7-8 Copy Operation Dialog



- a. Browse for the **From Variable** by expanding the tree until you find the client input variable name (as defined in the BPEL process WSDL file).
 - b. Browse for the **To Variable** by expanding the tree until you find the input variable name used by the web service (as defined in the message type definition of the web service WSDL).
8. Close the Create Copy Operation and Assign windows.
9. Save the file.

7.3.1.5 Creating a Second Assign Activity Inside a Scope Activity

This second assign activity returns a price quote in response to the client's purchase order.

To create a second assign activity inside a scope activity:

1. Drag and drop a second **Assign** activity below the **Invoke** activity and within the **Scope** activity.
2. Double-click the **Assign** icon.
3. Use the **General** tab of the Assign dialog to enter a name for the activity.
4. Click **Apply**.
5. Click the **Copy Operation** tab.
6. From the **Create** list, select **Copy Operation**.
7. Use the Create Copy Operation dialog to copy the contents of one variable to another.
 - a. Browse for the **From Variable** by expanding the tree until you find the client output variable name (as defined in the BPEL process WSDL file).
 - b. Browse for the **To Variable** by expanding the tree until you find the output variable name used by the web service (as defined in the message type definition of the web service WSDL).
8. Close the Create Copy Operation and Assign windows.
9. Save the file.

7.3.1.6 Creating a Receive Activity

A receive activity waits for a message from the client or a callback response message from an asynchronous web service.

To create a receive activity:

1. Drag and drop a **Receive** activity below the **Invoke** activity.
2. Double-click the **Receive** icon.
3. Use the **General** tab to enter values for the receive activity, as shown in [Figure 7-9](#).

Figure 7–9 Receive Dialog

The screenshot shows the 'Receive' dialog box with the following details:

- Title Bar:** Receive
- Tabs:** General (selected), Correlations, Sensors, Adapters, Annotations
- Name:** ReceiveSelectManufacturer
- My Role WebService Interface:**
 - Partner Link:** SelectService (with a flashlight icon for browsing)
 - Operation:** processRequestQuoteResponse (selected from a dropdown)
- Variable:** selectManufacturerResponse (with a flashlight icon for browsing)
- Create Instance:** ☐
- Buttons:** Help, Apply, OK, Cancel

- a. Enter a name for the **Receive** activity.
 - b. Use the flashlight icon to browse for the partner link of the web service.
This value is supplied by the web service WSDL.
 - c. Select an operation.
This value is supplied by the web service WSDL.
 - d. Use the flashlight icon to browse for the variable that holds the response from the web service.
This value is supplied by the web service WSDL.
4. Save the file.

7.3.2 What Happens When You Invoke a Web Service

For a synchronous web service invocation, the following takes place:

1. The **receiveInput** receive activity receives input from the user (client), as defined in the WSDL file (*bpel_process_name.wsdl*).
2. The first assign activity packages the data from the client so that it can be accepted by the web service.
3. The partner link sends the repackaged data to the web service and receives a response.
4. The second assign activity repackages the response into a **replyOutput** activity so that it can be accepted by the client application.
5. The **replyOutput** activity sends the repackaged response back to the client.

[Example 7–2](#) shows the partner link definition and invoke element for a synchronous web service.

Example 7–2 Source Code for the Partner Link Definition and Invoke Element of a Synchronous Web Service

```
<!-- This section shows where the partner link is defined.-->
  <partnerLink name="RapidService" partnerRole="RapidQuote_Role"
    partnerLinkType="ns23:RapidQuote_PL" />
...
  <scope name="CallRapidManufacturer">
    <variables>
      <variable name="manufacturerRequest"
        messageType="ns23:RequestQuotePortType_POItemsQuote" />
    </variables>
    <sequence name="Sequence_6">
      <assign name="TransformRapidRequest">
        <bpelx:annotation>
          <bpelx:pattern>transformation</bpelx:pattern>
        </bpelx:annotation>
        <copy>
          <from expression="ora:processXSLT('RapidTransformation.xsl',
            bpws:getVariableData('inputVariable','payload'))" />
          <to variable="manufacturerRequest" part="parameters" />
        </copy>
      </assign>
      <invoke name="InvokeRapidManufacturer"
        partnerLink="RapidService" portType="ns23:RequestQuote"
        operation="POItemsQuote"
        inputVariable="manufacturerRequest"
        outputVariable="rapidManufacturerResponse" />
    </sequence>
  </scope>
```

For an asynchronous web service invocation, the following takes place:

1. An assign activity prepares the purchase order.
2. An invoke activity (**InvokeSelectManufacturer**) initiates the request. The contents of this request are put into a request variable. The request variable is sent to the asynchronous web service.

When the request is initiated, a correlation ID unique to the client and partner link initiating the request is also sent to the web service. The correlation ID ensures that the correct response is returned to the corresponding requestor.

3. The web service sends the response to the receive activity (**ReceiveSelectManufacturer**), which is tracked by the correlation ID.
4. An assign activity (**AssignSelectManufacturer**) reads the response.

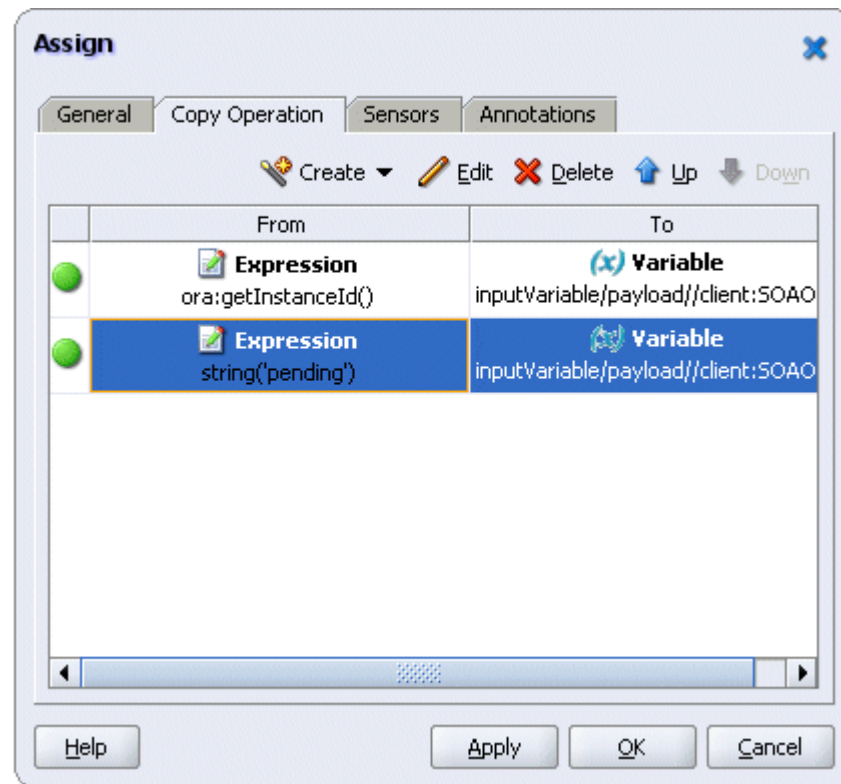
[Example 7–3](#) shows the source code for the partner link definition and the invoke and receive elements for an asynchronous web service.

Example 7-3 Source Code for the Partner Link Definition and Invoke and Receive Elements of an Asynchronous Web Service

```
<!-- This section shows where the partner link is defined.-->
<partnerLink myRole="SelectServiceRequester" name="SelectService"
    partnerRole="SelectServiceProvider"
    partnerLinkType="ns17:SelectService_PL"/>
...
<scope name="GetSelectMfrQuote">
    <variables>
        <variable name="manufacturerRequest"
            messageType="ns17:RequestQuote_processRequestQuote"/>
    </variables>
    <sequence name="Sequence_15">
        <assign name="TransformSelectRequest">
            <bpelx:annotation>
                <bpelx:pattern>transformation</bpelx:pattern>
            </bpelx:annotation>
            <copy>
                <from expression="ora:processXSLT('SelectTransformation.xml',bpws:getVariableData
                    ('inputVariable','payload'))"/>
                <to variable="manufacturerRequest" part="parameters"/>
            </copy>
        </assign>
        <invoke name="InvokeSelectManufacturer"
            partnerLink="SelectService"
            portType="ns17:SelectService"
            operation="processRequestQuote"
            inputVariable="manufacturerRequest"/>
        <receive name="ReceiveSelectManufacturer" createInstance="no"
            partnerLink="SelectService"
            portType="ns17:SelectServiceCallback"
            operation="processRequestQuoteResponse"
            variable="selectManufacturerResponse"/>
    </sequence>
</scope>
```

7.4 Using an Assign Activity

The starting point for data manipulation in BPEL is the assign activity, which builds on the XPath standard. As shown in [Figure 7-10](#), you use the assign activity to copy data from one XML variable to another, or to calculate the value of an expression and store it in a variable.

Figure 7–10 Assign Dialog for Copy Operations

A copy element within the activity specifies the source and target of the assignment (what to copy from and to), which must be of compatible types.

You can use XPath queries, expressions, and functions to manipulate data as follows:

- **XPath queries:** An XPath query selects a field within a source or target variable part. The `from` or `to` clause can include a query attribute whose value is an XPath query string. For example:

```
<from variable="input" part="payload"
      query="/p:CreditFlowRequest/p:ssn">
```

For XPath version 1.0, the value of the query attribute must be an absolute location path that selects exactly one node.

- **XPath expressions:** You use an XPath expression (specified in the `expression` attribute of the `from` clause) to indicate a value to be stored in a variable. For example:

```
<from expression="100"/>
```

The expression can be any general expression—that is, an XPath expression that evaluates to any XPath value type.

Within XPath expressions, you can call the following types of functions:

- Core XPath functions
- BPEL extension functions (`bpws:`)
- BPEL XPath extension functions (`ora:`)
- Custom functions

See *Oracle BPEL Process Manager Developer's Guide* for more information about these functions.

Sophisticated data manipulation can be difficult to perform with the BPEL assign activity and the core XPath functions. However, you can perform complex data manipulation and transformation by using XSLT or Java, or as a web service. For XSLT, Oracle BPEL Process Manager includes XPath functions that execute these transformations. See the following for information about transformations:

- [Section 7.5, "Creating a Transformation"](#)
- *Oracle BPEL Process Manager Developer's Guide*

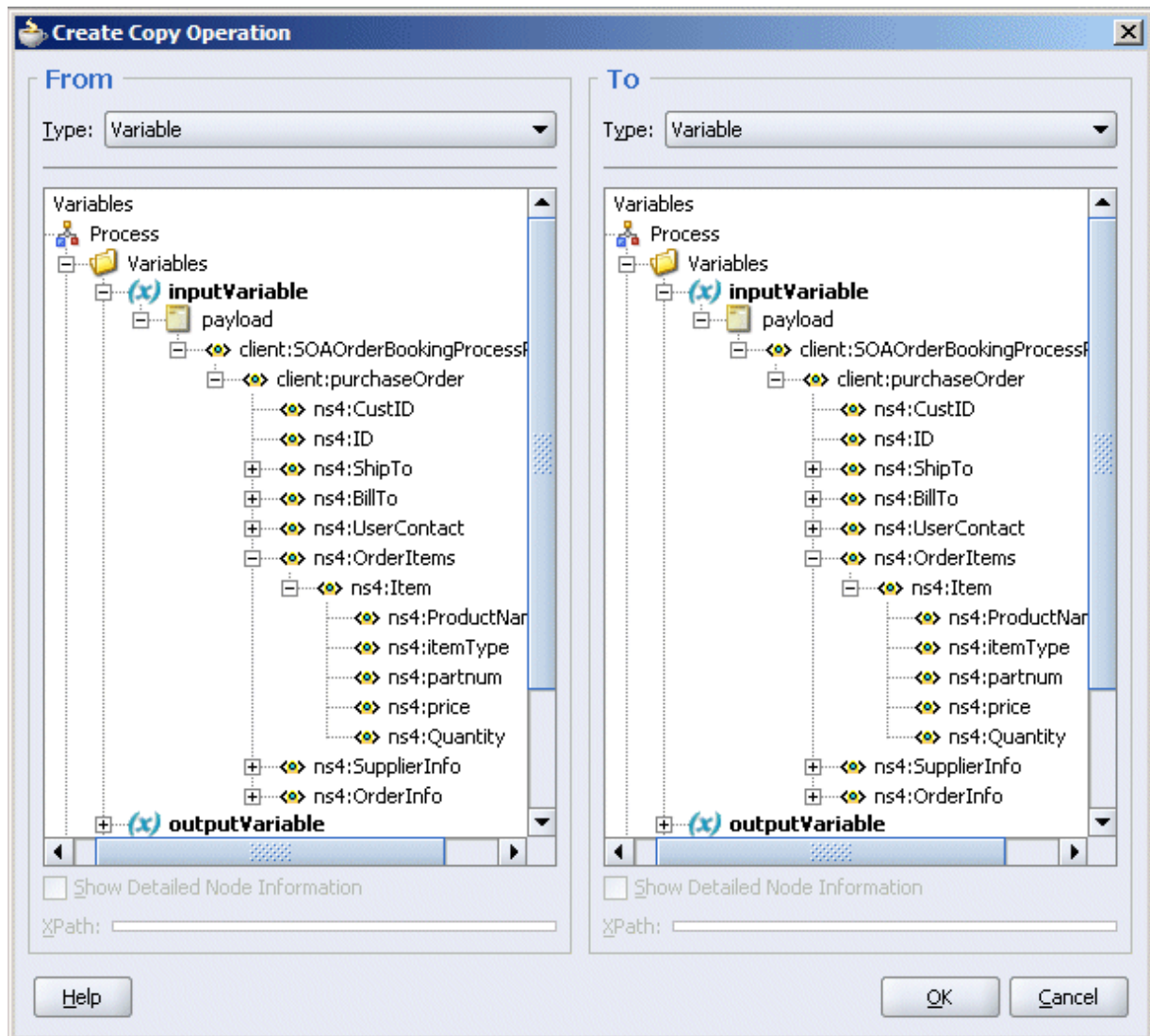
7.4.1 How to Use an Assign Activity

You use the copy operation within the assign activity to specify which data is being copied into which variable, as shown in [Figure 7–10](#). In the SOA Order Booking application, the BPEL process uses an assign activity, **InitializeRequest**, (within the **CreditService** scope) to copy the value of the `creditcardtype` variable to the `ccType` variable, and to copy the value of the `creditcardnumber` variable to the value of the `ccNum` variable. The variable names `ccType` and `ccNum` are specified in the WSDL of the **CreditValidatingService** partner link.

To copy variables using an assign activity:

1. Drag and drop an **Assign** activity inside a **Scope** activity.
2. Double-click the **Assign** icon.
3. Use the **General** tab of the Assign dialog to enter a name for the **Assign** activity.
4. Click **Apply**.
5. Click the **Copy Operation** tab.
6. From the **Create** list, select **Copy Operation**.
7. Use the Create Copy Operation dialog, shown in [Figure 7–8](#), to copy the contents of one variable to another. This **Assign** activity takes the client's purchase order as input and contacts the Rapid Manufacturer web service for a price quote.

Figure 7-11 Copy Operation Dialog



- a. Browse for the **From Variable** by expanding the tree until you find the client input variable name (as defined in the BPEL process WSDL file).
- b. Browse for the **To Variable** by expanding the tree until you find the input variable name used by the web service (as defined in the message type definition of the web service WSDL).
8. Close the Create Copy Operation and Assign windows.
9. Save the file.

7.4.2 What Happens When You Use an Assign Activity

[Example 7–4](#) shows the source code for the **InitializeRequest** assign activity in the SOA Order Booking application.

Example 7–4 Source Code for the *InitializeRequest* Assign Activity

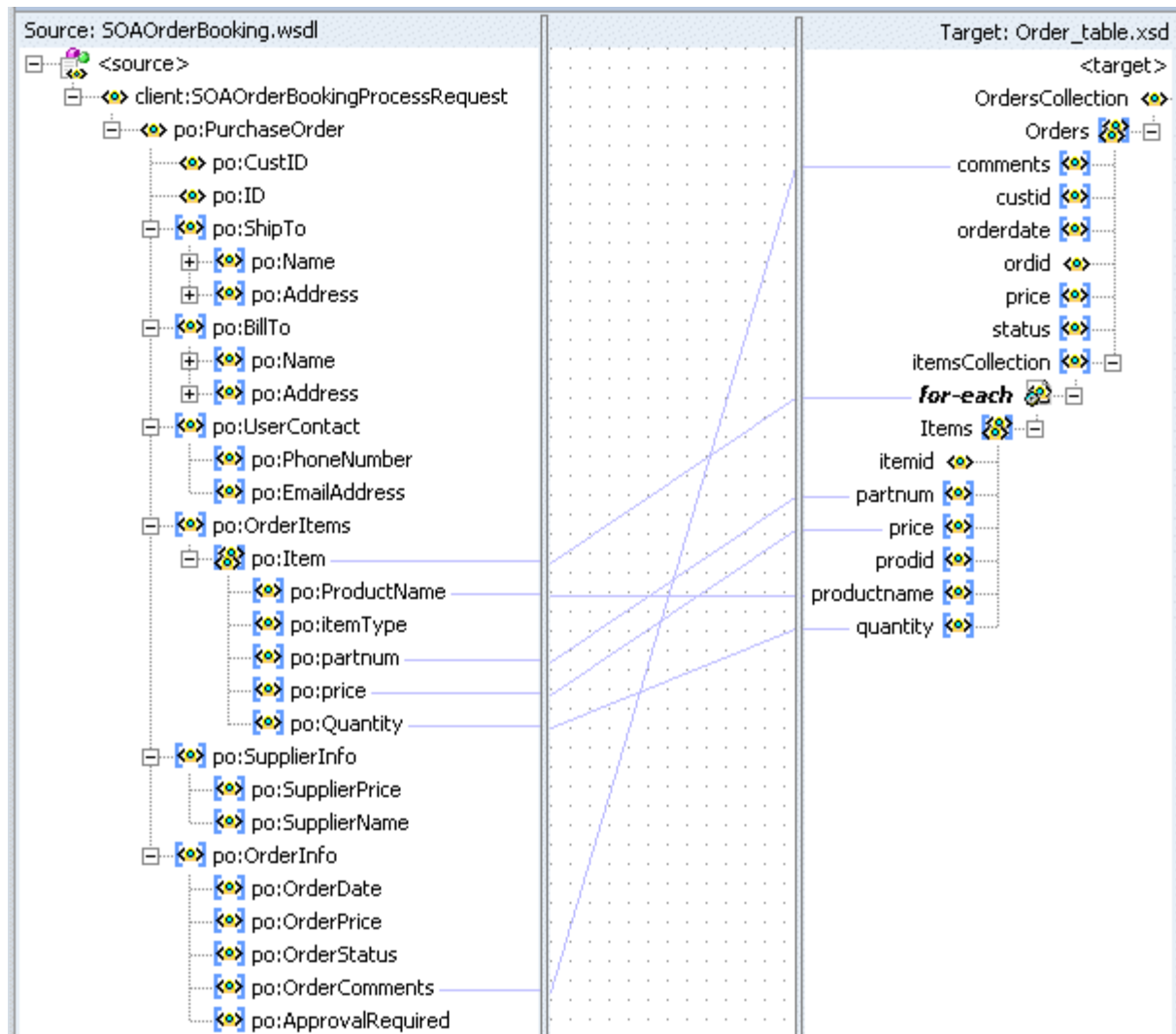
```
<scope name="CreditService">
  <variables>
    <variable name="validateRequest"
      messageType="ns6:CreditCardValidationRequestMessage"/>
    <variable name="validateResponse"
      messageType="ns6:CreditCardValidationResponseMessage"/>
  </variables>
  <sequence name="Sequence_5">
    <assign name="InitializeRequest">
      <copy>
        <from variable="customerServiceResponse"
          query="/ns3:findCustomerByIdResponse/ns3:return/ns3:creditcardtype"
          part="parameters"/>
        <to variable="validateRequest" part="CreditCard"
          query="/ns21:CreditCard/ccType"/>
      </copy>
      <copy>
        <from variable="customerServiceResponse"
          query="/ns3:findCustomerByIdResponse/ns3:return/ns3:creditcardnumber"
          part="parameters"/>
        <to variable="validateRequest" part="CreditCard"
          query="/ns21:CreditCard/ccNum"/>
      </copy>
    </assign>
  </sequence>
</scope>
```

7.5 Creating a Transformation

A transformation maps the elements of a source schema to the elements of a target schema. Because a web service typically specifies the format required for incoming data, for example, the format of an invoice, you need a transformation to map the source data schema, for example, a purchase order, to the target data schema.

You use the transform activity to add transformation logic to a BPEL process. The transformation defines the source and target variables, the message parts to map, and the XSL map file. To specify the source and target schema mapping, you use the XSLT Mapper, shown in [Figure 7–12](#).

Figure 7-12 The XSLT Mapper



The source and target schemas that you identify when creating the XSL map are displayed in the XSLT Mapper. To copy an attribute or leaf-element in the source to an attribute or leaf-element in the target, drag and drop the source to the target. To do more complex mappings, drag and drop functions into the mapper canvas (in the center) and connect the functions to the source and target nodes. From the Component Palette, the following mapping functionality is available:

- **Advanced functions:** Examples of these functions are `lookup-xml`, which looks up values in XML files, and `xpath-expression`, with which you can build an XPath expression into the mapping.
- **Code snippets:** Enables you to add Java code to the mapping.
- **Conversion functions:** Examples are `boolean`, which converts a parameter to a Boolean, and `string`, which converts a parameter to a string.
- **Cumulative functions:** Enables you to count or sum the nodes in a node set.
- **Database functions:** Enables you to query a database.

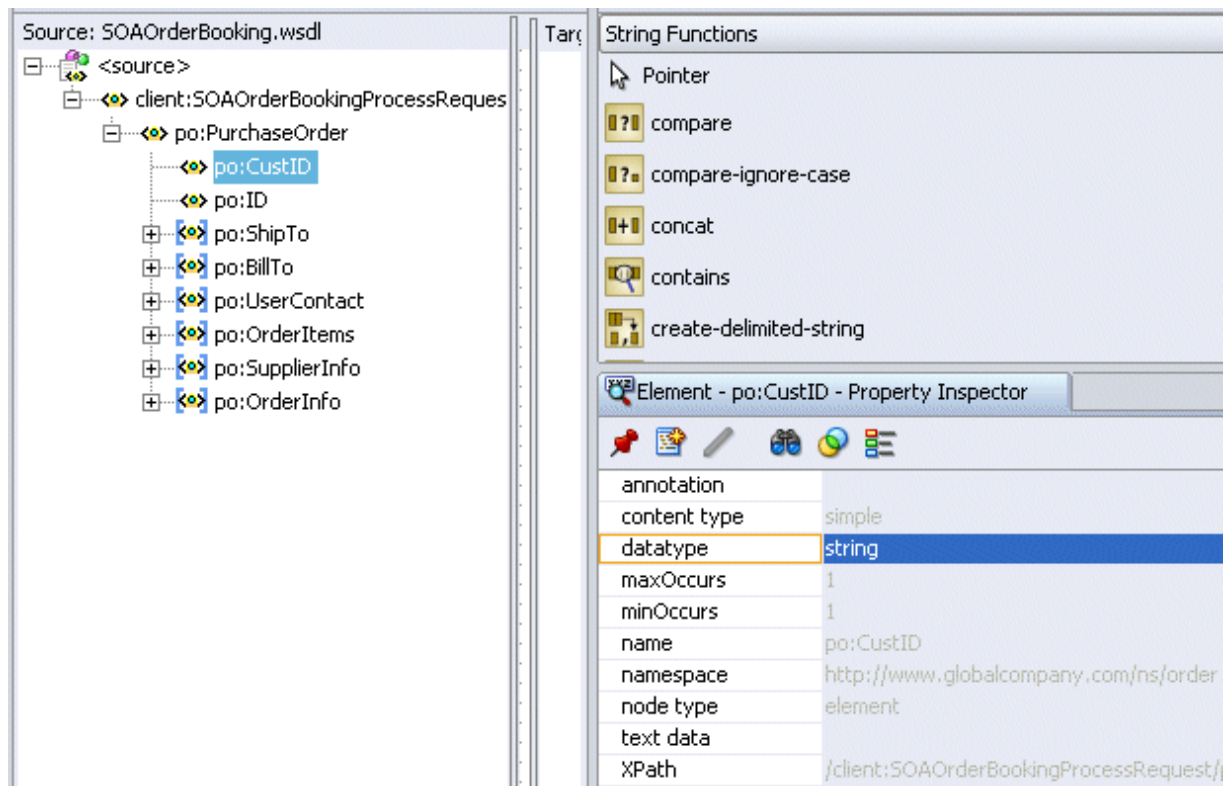
- Date functions: Examples include `current-date`, which returns the current date in ISO format, and `implicit-timezone` which returns the current time zone in ISO format.
- Logical functions: Examples include `logical and` and `or`, and the functions for comparing parameters, such as `greater` and `less`.
- Mathematical functions: Examples include `abs`, which returns the absolute value of the input number, and arithmetic operations such as `add`, `subtract`, `multiply`, and `divide`.
- Node-set functions: Examples include `current`, which returns a node set that has the current node as its only member, and `document`, which provides access to XML documents other than the main source document.
- String functions: Examples include `concat`, which returns the concatenation of two strings (for example, concatenate `first` and `last` name elements and map them to a target `fullname` element), and `lower-case`, which returns an input string with all characters converted to their lower-case equivalents.
- User-defined extension functions: You can import your own set of Java functions. From the **Tools** menu, select **Preferences**, and then **XSL Maps**.
- User-defined named templates: For a complex mapping, you can define a template in the XSLT language, which you then use like a function.
- XML: You can manipulate the XML content, or get information about it.
- XSLT constructs: Examples include `xsl:if`, which enables you to apply a condition in your map (for example, map a source to a target only if it exists), and `for-each`, which maps repeating elements in the source to repeating elements in the target (useful for handling arrays).
- WC3-recommended XSLTs: Examples include `xsl:element`, in which an element is created with a computed name, and `xsl:attribute`, in which an element is used to add attributes to result elements. For a description of the XSLTs recommended by the World Wide Web Consortium, go to

www.w3.org/1999/Transform

BPPEL processes and ESB projects can use the same XSLT.

The Property Inspector, shown in [Figure 7-13](#), displays the properties of a selected XSD element.

Figure 7-13 Property Inspector



See *Oracle BPEL Process Manager Developer's Guide* for information on the following transformation and XSLT Mapper topics:

- Examples using the XSLT Mapper and context-sensitive menus, including how to set constant values, use extension functions, edit function parameters, chain functions, and edit XPath expressions
- How to create XSL map files without using the XSLT Mapper
- How to map nodes automatically using the **Auto Map Preferences** feature
- How to generate mapping dictionaries (synonyms for mappings) using the **Generate Dictionary** feature and HTML reports using the **Generate Report** feature
- How to test a map

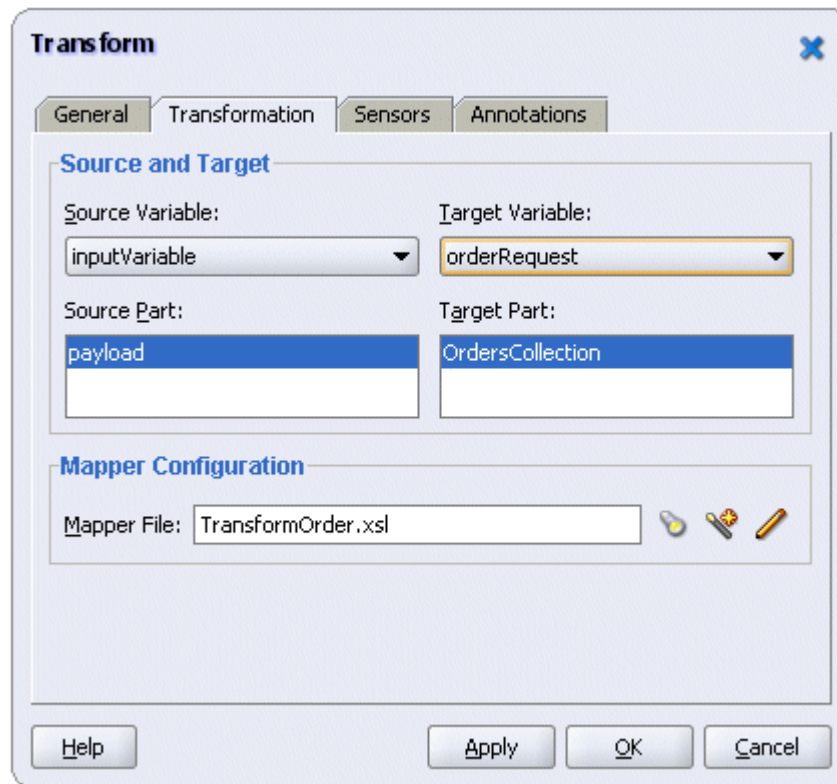
7.5.1 How to Create a Transformation

To create a transformation, use a transform activity and the XSLT Mapper, which assigns elements of a source schema to the elements of a target schema.

To create a transformation:

1. Drag and drop a **Transform** activity into the BPEL process.
2. Double-click the **Transform** activity.
3. Use the Transform dialog (shown in [Figure 7-14](#)) to enter source and target schema information.

Figure 7–14 Creating a Transformation



- a. Select the source variable from which to map elements.
- b. Select the source part of the variable (for example, a payload schema consisting of a purchase order request) from which to map.
- c. Select the target variable where elements are mapped to.
- d. Select the target part of the variable (for example, a payload schema consisting of an invoice) to which to map.
- e. Enter a mapper file name or accept the default name (or use the flashlight icon to browse for a file).

This file stores the mapping you create using the XSLT Mapper transformation tool.

- f. Click the **Create Mapping** icon.
- g. Use the XSLT Mapper to drag and drop from source attributes or leaf-elements to target attributes or leaf-elements, as shown in [Figure 7–12](#). Use the functions in the Component Palette and the automap feature to simplify this task.

You can also create an XSL map file from an XSL stylesheet. From the **File** main menu in Oracle JDeveloper, click **New**, then **XML**, and then **XSL Map From XSL Stylesheet**.

See *Oracle BPEL Process Manager Order Booking Tutorial* for step-by-step instructions on dragging from source to target in the XSLT Mapper.

7.5.2 What Happens When You Create a Transformation

When you use the XSLT mapper, the transformation is captured in an XSL map file. [Example 7–5](#) shows the source code for `TransformOrder.xsl`, which maps the order schema in `SOAOrderBooking.wsdl` to the database schema required by the Order web service.

Example 7–5 Source Code for `TransformOrder.xsl`

```
<?xml version="1.0" encoding="UTF-8" ?>
<?oracle-xsl-mapper
  <!-- SPECIFICATION OF MAP SOURCES AND TARGETS, DO NOT MODIFY. -->
  <mapSources>
    <source type="WSDL">
      <schema location="SOAOrderBooking.wsdl"/>
      <rootElement name="SOAOrderBookingProcessRequest" namespace="http://
        www.globalcompany.com/ns/OrderBooking"/>
    </source>
  </mapSources>
  <mapTargets>
    <target type="XSD">
      <schema location="Order_table.xsd"/>
      <rootElement name="OrdersCollection" namespace="http://
        xmlns.oracle.com/pcbpel/adapter/db/top/Order"/>
    </target>
  </mapTargets>
  <!-- GENERATED BY ORACLE XSL MAPPER 10.1.3.1.0(build 060515.0101) AT [SAT JUL 08 16:58:22 PDT
2006]. -->
  ?>
  <xsl:stylesheet version="1.0"
    xmlns:bpws="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
    xmlns:plnk="http://schemas.xmlsoap.org/ws/2003/05/partner-link/"
    xmlns:ns1="http://xmlns.oracle.com/SOAOrderBooking"
    xmlns:ns4="http://www.globalcompany.com/ns/OrderBooking/rules"
    xmlns:ns0="http://www.w3.org/2001/XMLSchema"
    xmlns:hxf="http://xmlns.oracle.com/bpel/workflow/xpath"
    xmlns:xp20="http://www.oracle.com/XSL/Transform/java/
      oracle.tip.pc.services.functions.Xpath20"
    xmlns:client="http://www.globalcompany.com/ns/OrderBooking"
    xmlns:ns2="http://xmlns.oracle.com/pcbpel/adapter/db/top/Order"
    xmlns:cu="http://www.globalcompany.com/ns/customer"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:po="http://www.globalcompany.com/ns/order"
    xmlns:wf="http://schemas.oracle.com/xpath/extension"
    xmlns:ids="http://xmlns.oracle.com/bpel/services/IdentityService/xpath"
    xmlns:orcl="http://www.oracle.com/XSL/Transform/java/
      oracle.tip.pc.services.functions.ExtFunc"
    exclude-result-prefixes="xsl plnk ns4 ns0 client po ns2 bpws hxf xp20 wf ids orcl">
  <xsl:template match="/">
    <ns3:OrdersCollection>
      <ns3:Orders>
        <ns3:comments>
          <xsl:value-of select="/client:SOAOrderBookingProcessRequest/po:PurchaseOrder/
            po:OrderInfo/po:OrderComments"/>
        </ns3:comments>
      </ns3:Orders>
    </ns3:OrdersCollection>
  </template>
</xsl:stylesheet>
```

```
<ns3:itemsCollection>
  <xsl:for-each select="/client:SOAOrderBookingProcessRequest/po:PurchaseOrder/
    po:OrderItems/po:Item">
    <ns3:Items>
      <ns3:partnum>
        <xsl:value-of select="po:partnum"/>
      </ns3:partnum>
      <ns3:price>
        <xsl:value-of select="po:price"/>
      </ns3:price>
      <ns3:productname>
        <xsl:value-of select="po:ProductName"/>
      </ns3:productname>
      <ns3:quantity>
        <xsl:value-of select="po:Quantity"/>
      </ns3:quantity>
    </ns3:Items>
  </xsl:for-each>
</ns3:itemsCollection>
</ns3:Orders>
</ns3:OrdersCollection>
</xsl:template>
</xsl:stylesheet>
```

7.6 Communicating with a Database

For Oracle BPEL Process Manager and a database to communicate, the database is exposed as a web service. As with other web services, this communication is defined in a partner link. However, to give a BPEL process access to database tables, you also need to configure an adapter. The Adapter Configuration Wizard feature of Oracle BPEL Process Manager enables you to integrate BPEL processes with database tables, as well as file systems, database queues, Java Message Services (JMS), and Oracle Applications. After you create the partner link and define a database adapter service, a WSDL file that includes the adapter configuration settings is created.

The database adapter connects to any relational database. For nonrelational databases and legacy systems, application and mainframe adapters are available. See *Oracle Application Server Adapter Concepts* for information about application and mainframe adapters.

To access an existing relational schema, you use the Adapter Configuration Wizard to do the following:

- Import a relational schema and map it as an XML schema (XSD)
- Abstract SQL operations such as `SELECT`, `INSERT`, and `UPDATE` as web services

While your BPEL process deals with XML and invokes web services, database rows and values are queried, inserted, and updated.

Features of the database adapter include:

- Compliance with open standards. The database adapter is an implementation of J2EE Connector Architecture (J2CA) 1.5 standards and is deployed as a resource adapter in the same Oracle Application Server Containers for J2EE (OC4J) container as Oracle BPEL Process Manager.
- Connectivity to any relational (SQL 92) database using JDBC, or ODBC using the Sun JdbcOdbcBridge

- Ability to map any existing relational schema to XML. The mapping is nonintrusive to the schema and no changes need to be made to it.
- Web services abstraction of SQL operations. The generated WSDL operations are `merge`, `insert`, `update`, `write`, `delete`, `select`, `queryByExample`, and inbound polling, which includes physical delete, logical delete, and sequencing-based polling strategies.
- Leveraging of Oracle TopLink technology, an advanced object-to-relational persistence framework. You can access the underlying TopLink project, and use the TopLink Workbench interface for advanced mapping and configuration, sequencing, batch and joined relationship reading, batch writing, parameter binding, statement caching, connection pooling, external transaction control (JTS and JTA), `UnitOfWork` for minimal updates, caching, optimistic locking, advanced query support, and query by example.

See *Oracle Application Server Adapter for Files, FTP, Databases, and Enterprise Messaging User's Guide* for more information.

7.6.1 How to Configure a Database Adapter

Use the Adapter Configuration Wizard to configure a database adapter. With the wizard, you can import tables from the database, specify relationships spanning multiple tables, generate corresponding XML schema definitions, and create services to expose the necessary SQL or database operations. These services are consumed to define partner links that are used in the BPEL process. You use the Adapter Configuration Wizard to both create and edit adapter services.

To configure a database adapter:

1. Drag and drop a **Database Adapter** service into one of the **Services** swim lanes.
2. Enter a service name.
3. Use the Adapter Configuration Wizard - Service Connection page to provide a database connection and a Java Naming and Directory Interface (JNDI) name, or accept the default JNDI name.

The JNDI name acts as a placeholder for the connection used when your service is deployed to Oracle BPEL Server. This enables you to use different databases for development and production. The Adapter Configuration Wizard captures the design-time connection in the generated WSDL as well, to serve as a fallback in case the runtime lookup fails.

4. Use the Adapter Configuration Wizard - Operation Type page to select an operation type.
 - **Call a Stored Procedure or Function:** Select this option if you want the service to execute a stored procedure or function.
 - **Perform an Operation on a Table:** Select this option for outbound operations. You can select **Insert or Update**, **Delete**, **Select**, or any combination of the three. These operations loosely translate to SQL `INSERT`, `UPDATE`, `DELETE`, and `SELECT` operations. If you select all three, then after you run the wizard, you see the following operations in the **Operation** list of the Invoke window: **merge**, **insert**, **update**, **write**, **delete**, **serviceNameSelect**, and **queryByExample**.

- **Poll for New or Changed Records in a Table:** Select this option for an inbound operation (that is, an operation that is associated with a receive activity). This operation type polls a specified table and returns for processing any new rows that are added. You can also specify the polling frequency.
- 5. Use the Adapter Configuration Wizard - Select Table page to select already imported tables or to import tables.

Select the root database table for your operation. If you are using multiple, related tables, then this is the highest-level table (or highest parent table) in the relationship tree. All the tables that have been previously imported in the project (including tables that were imported for other partner links) are displayed. This enables you to reuse configured table definitions across multiple partner links in a given BPEL project. These are the generated TopLink descriptors.

If you are importing tables, click **Import Table**, select a schema, click **Query**, and move the tables you want from the **Available** list to the **Selected** list.

If you want to reimport a table (if the table structure has changed on the database, for example), import it again. This overwrites the previously configured table definition. If you reimport a table, you lose any custom relationships you may have defined on that table, as well as any custom WHERE clauses (if the table being imported was the root table).

- 6. Click **Next**.
- 7. Use the Adapter Configuration Wizard - Define Primary Key page to define primary keys for any of the tables you selected in the previous step.

This page appears if primary keys are not already defined. You must specify a primary key for all imported tables. You can select multiple fields if you need to specify a multipart primary key. The primary key that you specify here is recorded in the offline database table and is not persisted back to the database schema; the database schema is left untouched.
- 8. Use the Adapter Configuration Wizard - Relationships page to create a new relationship between two tables or to remove relationships between tables.
 - a. Click **Create**.
 - b. Select the parent and child tables.
 - c. Select the mapping type (one-to-many, one-to-one, or one-to-one with the foreign key on the child table).
 - d. Associate the foreign key fields to the primary key fields.
 - e. Name the relationship or accept the default name.

Note: Only tables that are reachable from the root table can be selected as a parent.

- 9. Use the Adapter Configuration Wizard - Object Filtering page to review the object model.

10. Use the Adapter Configuration Wizard - Define Selection Criteria page to further define a SQL query.
 - a. In the **Parameters** section, click **Add** to add a parameter (for an outbound operation only). To remove a parameter, highlight the parameter and click **Remove**.
 - b. In the **SQL** section, edit the SQL syntax for an outbound operation (inbound operations syntax is read-only.) Or click **Edit** to use the Expression Builder to edit both outbound and inbound operations.
 - c. Use the **Return a Single Result Set** option to query multiple related tables with a single SQL `SELECT` statement and return a single result set.

See *Oracle Application Server Adapter for Files, FTP, Databases, and Enterprise Messaging User's Guide* for information on the following topics:

- SQL operations as web services
- Polling strategies
- Creating relationships
- Stored procedures and functions
- Performance issues related to returning a single result set
- Other adapters--AQ, File, FTP, JMS, MQ, Oracle Applications, and so on

7.6.2 What Happens When You Configure a Database Adapter

When you import a table, the offline table support of Oracle JDeveloper BPEL Designer creates an offline snapshot of the database table. You can modify this offline version of the table (for example, you can add a foreign key constraint) without affecting the real database table. This creates a TopLink descriptor and associated Java source file for the table, and all the attributes in the descriptor are automapped to their corresponding database columns. The TopLink descriptor maps the Java class to the offline database table.

Most typical data columns are mapped as direct-to-field mappings, meaning that the value in the database column is directly mapped to the attribute. For example, a `SALARY` column in the database is mapped to a `salary` attribute in the object model, and that attribute contains the value of that column.

If foreign key constraints are already present in the imported tables, then relationship mappings are autogenerated between the tables. To cover as many scenarios as possible, two mappings are generated for every foreign key constraint encountered: a one-to-one mapping from the source table to the target table, and a one-to-many mapping in the opposite direction. After this is done, you have an Oracle TopLink Workbench project in your BPEL project.

Note: The Java classes that are created as part of the descriptor generation process are never actually deployed with your process or used at runtime. They are present in the design time because Oracle TopLink Workbench is expecting each descriptor to be associated with a Java class. When your process is deployed, the mapping metadata is stored in `toplink_mappings.xml`.

When you have finished importing tables, you must select a root database table. In doing so, you are actually selecting which TopLink descriptor stores the autogenerated query.

At design time, the following files are generated:

- *service_name.wsdl*: Contains the database adapter service definition
- *RootTable.xsd*: The XML type definition of the root object
- *toplink_mappings.xml*: Contains the TopLink mapping metadata for your BPEL project. It is the only Toplink artifact that is deployed to the server.

[Example 7–6](#) shows the source code for the database adapter WSDL file, *Order.wsdl*, which is used to write order information to a database in the SOA Order Booking application.

Example 7–6 Source Code for Order.wsdl

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions
  name="Order"
  targetNamespace="http://xmlns.oracle.com/pcbpel/adapter/db/Order/"
  xmlns:tns="http://xmlns.oracle.com/pcbpel/adapter/db/Order/"
  xmlns:plt="http://schemas.xmlsoap.org/ws/2003/05/partner-link/"
  xmlns:jca="http://xmlns.oracle.com/pcbpel/wsdl/jca/"
  xmlns:pc="http://xmlns.oracle.com/pcbpel/"
  xmlns:top="http://xmlns.oracle.com/pcbpel/adapter/db/top/Order"
  xmlns:hdr="http://xmlns.oracle.com/pcbpel/adapter/db/"
  xmlns="http://schemas.xmlsoap.org/wsdl/">
  <import namespace="http://xmlns.oracle.com/pcbpel/adapter/db/"
    location="DBAdapterOutboundHeader.wsdl" />
  <types>
    <schema xmlns="http://www.w3.org/2001/XMLSchema">
      <import namespace="http://xmlns.oracle.com/pcbpel/adapter/db/top/Order"
        schemaLocation="Order_table.xsd" />
    </schema>
  </types>
  <message name="OrdersCollection_msg">
    <part name="OrdersCollection" element="top:OrdersCollection"/>
  </message>
  <portType name="Order_ptt">
    <operation name="merge">
      <input message="tns:OrdersCollection_msg"/>
    </operation>
    <operation name="write">
      <input message="tns:OrdersCollection_msg"/>
    </operation>
  </portType>
```



```

<binding name="Order_binding" type="tns:Order_ptt">
  <jca:binding />
  <operation name="merge">
    <jca:operation
      InteractionSpec="oracle.tip.adapter.db.DBWriteInteractionSpec"
      DescriptorName="Order.Orders"
      DmlType="merge"
      DetectOmissions="true"
      OptimizeMerge="true"
      MappingsMetaDataURL="Order_toplink_mappings.xml" />
    <input>
      <jca:header message="hdr:OutboundHeader_msg" part="outboundHeader" />
    </input>
  </operation>
  <operation name="write">
    <jca:operation
      InteractionSpec="oracle.tip.adapter.db.DBWriteInteractionSpec"
      DescriptorName="Order.Orders"
      DmlType="write"
      MappingsMetaDataURL="Order_toplink_mappings.xml" />
    <input>
      <jca:header message="hdr:OutboundHeader_msg" part="outboundHeader" />
    </input>
  </operation>
</binding>
<!-- Your runtime connection is declared in
J2EE_HOME/application-deployments/default/DbAdapter/oc4j-ra.xml
These 'mcf' properties here are from your design time connection and save you from having
to edit that file and restart the application server if eis/DB/SOADemo is missing.
These 'mcf' properties are safe to remove.
-->
<service name="Order">
  <port name="Order_pt" binding="tns:Order_binding">
    <jca:address location="eis/DB/soademo"
      UIConnectionName="SOADemo"
    />
  </port>
</service>
<plt:partnerLinkType name="Order_plt" >
  <plt:role name="Order_role" >
    <plt:portType name="tns:Order_ptt" />
  </plt:role>
</plt:partnerLinkType>
</definitions>

```

7.7 Creating Conditional Branching Using a Switch Activity

Conditional branching introduces decision points to control the flow of execution of a BPEL process. You can use a switch activity to produce different actions based on conditional branching. You set up two or more branches, with each branch in the form of an XPath expression. If the expression is true, then the branch is executed. If the expression is false, then the BPEL process moves to the next branch condition, until it either finds a valid branch condition, encounters an otherwise branch, or runs out of branches. If more than one branch condition is true, then BPEL executes the first true branch.

[Figure 7-15](#) shows a conditional branch from the SOA Order Booking application. The **SelectByPrice** switch activity is used to select the supplier that provides the lowest quote.

Figure 7-15 Conditional Branching with a Switch Activity

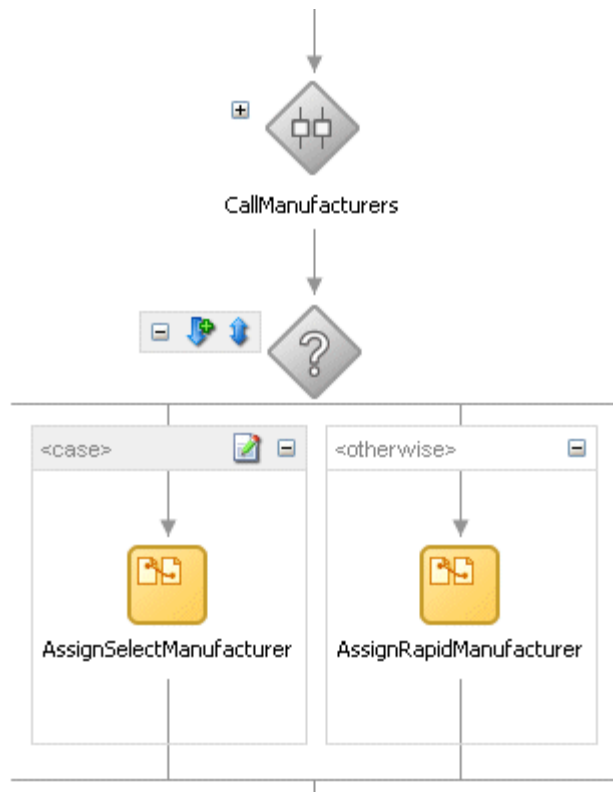
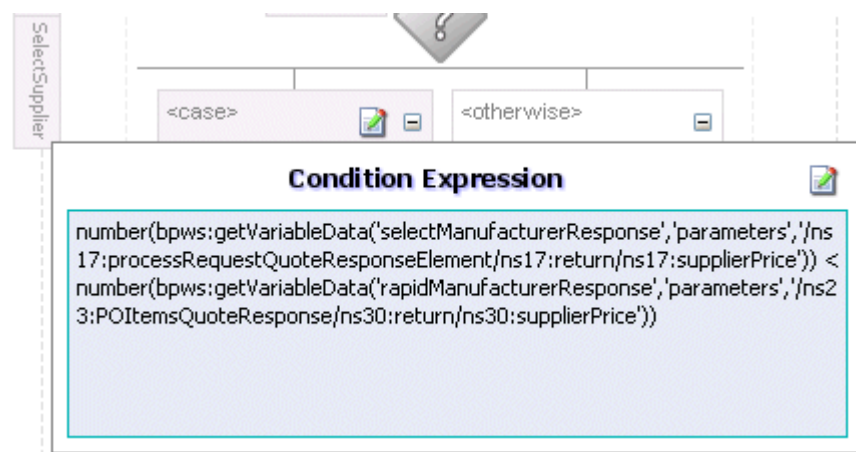


Figure 7-16 shows the XPath expression that defines the case branch. Select Manufacturer is selected if its price quote is less than Rapid Manufacturer's price quote.

Figure 7-16 The Case Branch of the `SelectByPrice` Switch Activity



If the case branch evaluates to false, then Rapid Manufacturer is selected.

See *Oracle BPEL Process Manager Developer's Guide* for information on the following conditional branching topics:

- Using a while activity to create a while loop to select between two actions
- Adding events and timeouts to a conditional branch

7.7.1 How to Create Conditional Branching Using a Switch Activity

To add conditional branching to a BPEL process, use a switch activity.

To create conditional branching using a switch activity:

1. Drag and drop a **Switch** activity into a BPEL flow.
2. Double-click the **Switch_1** icon to enter a name for the activity.
3. Expand the **Switch** activity to see the switch cases.
The **Switch** activity has two switch case branches by default, each with a box for functional elements. To add more branches, right-click the **Switch** icon and select **Add Switch Case** or **Add Switch Otherwise**.
4. Double-click <case> in the first branch.
5. Use the Switch Case dialog to specify the conditional branching logic, as shown in [Figure 7-17](#).

Figure 7-17 *Creating the Case Branch in the SelectByPrice Switch Activity*



- a. Enter a name for the case in the **Name** field.
- b. Click the notepad icon to use the XPath Expression Builder to create an XPath Boolean expression in the **Expression** field.
- c. Click **OK** to close both the Expression Builder and the SwitchCase dialogs.

7.7.2 What Happens When You Create Conditional Branching Using a Switch Activity

[Example 7–7](#) shows the source code for the **SelectByPrice** switch activity in the SOA Order Booking application.

Example 7–7 Source Code for the SelectByPrice Switch

```
<switch name="SelectByPrice">
  <!--case condition="number(bpws:getVariableData('selectManufacturerResponse',
    'parameters', '/ns22:processRequestQuoteResponseElement/ns22:result/
    ns22:supplierPrice')) &lt; number(bpws:getVariableData
    ('rapidManufacturerResponse', 'parameters', '/ns23:OrderQuoteResponse/
    ns24:return/ns25:supplierPrice'))"-->
  <case condition="true()">
    <assign name="AssignSelectManufacturer">
      <copy>
        <from variable="selectManufacturerResponse" part="parameters" query="/
          ns17:processRequestQuoteResponseElement/ns17:return/ns17:supplierPrice"/>
        <to variable="inputVariable" part="payload" query="/
          client:SOAOrderBookingProcessRequest/
          client:purchaseOrder/ns4:SupplierInfo/ns4:SupplierPrice"/>
      </copy>
      <copy>
        <from variable="selectManufacturerResponse" part="parameters" query="/
          ns17:processRequestQuoteResponseElement/ns17:return/ns17:supplierName"/>
        <to variable="inputVariable" part="payload" query="/
          client:SOAOrderBookingProcessRequest/client:purchaseOrder/ns4:SupplierInfo/
          ns4:SupplierName"/>
      </copy>
    </assign>
  </case>
  <otherwise>
    <assign name="AssignRapidManufacturer">
      <copy>
        <from variable="rapidManufacturerResponse" part="parameters" query="/
          ns23:POItemsQuoteResponse/ns30:return/ns30:supplierPrice"/>
        <to variable="inputVariable" part="payload" query="/
          client:SOAOrderBookingProcessRequest/client:purchaseOrder/ns4:SupplierInfo/
          ns4:SupplierPrice"/>
      </copy>
      <copy>
        <from variable="rapidManufacturerResponse" part="parameters" query="/
          ns23:POItemsQuoteResponse/ns30:return/ns30:supplierName"/>
        <to variable="inputVariable" part="payload" query="/
          client:SOAOrderBookingProcessRequest/client:purchaseOrder/ns4:SupplierInfo/
          ns4:SupplierName"/>
      </copy>
    </assign>
  </otherwise>
</switch>
```

7.8 Using Business Rules and a Decide Activity

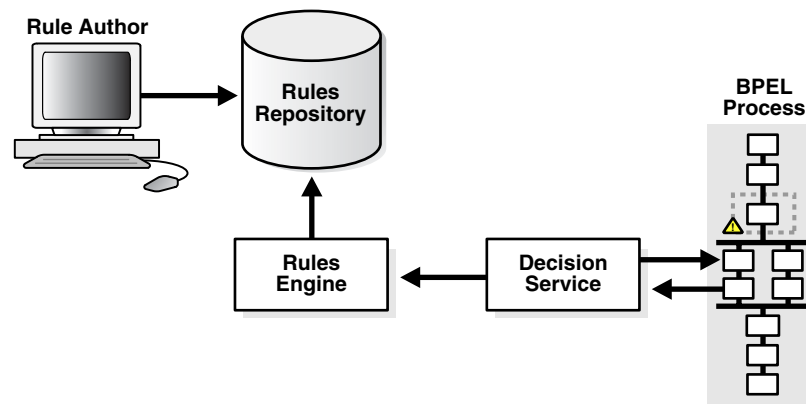
Business rules are statements that describe the policies of a company. When Oracle BPEL Process Manager is used in combination with a business rules engine such as Oracle Business Rules, you can effect changes in a BPEL process that are related to a company's policies without modifying, stopping, or redeploying the BPEL process itself. Moreover, a business analyst or anyone familiar with the policies of a company can change business rules without involving BPEL developers.

After business rules are defined using Oracle Business Rules, you add the following to implement the rules within a BPEL process:

- **Decision service partner link:** As with other partner links, the decision service creates a WSDL file, but it also starts the Decision Service Wizard. The Decision Service Wizard guides you through the selection of a rule set or function and the invocation pattern. Your selections are then converted into a web service that communicates with the BPEL process.
- **Decide activity:** This activity invokes the decision service partner link that you created with the Decision Service Wizard. You also use the decide activity to create copy operation assignments between the fact data in the rule set or function and the BPEL variables.

The process of integrating business rules with the preceding components of Oracle BPEL Process Manager is shown in [Figure 7–18](#).

Figure 7–18 Integrating Business Rules with Oracle BPEL Process Manager Components



In the SOA Order Booking application, the BPEL process includes a decide activity that works with a rule that implements the company's policy. Manual credit approval is required for all orders over \$1000 unless the customer has platinum status.

See the following for more information about using business rules:

- [Chapter 8, "Creating and Using Rules for a Decision Service"](#)
- *Oracle BPEL Process Manager Developer's Guide*
- *Oracle Business Rules User's Guide*

7.8.1 How to Use the Decision Service Wizard to Create a Decision Service Partner Link

Use the Decision Service Wizard to select a rule set or function that you created in the Oracle Business Rules Rules Engine. You also select the invocation pattern that you want performed from the following options:

- **Assert facts only:** Select the rule engine facts you want to assert (send factual data to the rules engine) in the future. You assign the required data for the facts with a BPEL assign activity. The underlying rule session must be stateful. Otherwise, the asserted facts are not visible to subsequent rule engine invocations.

- **Retrieve results:** Retrieve a result from the business rules engine. The values of these results may have changed by past execution of a rule set acting on these facts. The wizard assumes that it has a stateful rule session in its cache from which a result can be retrieved. This is the case if the invocation pattern **Assert facts and execute rule set** was executed previously in the BPEL process.
- **Assert facts and execute rule set:** The same as **Assert facts only**, except that the rule set is executed after the facts are asserted. The wizard creates (or uses) a stateful rule session. Otherwise, the result of executing this pattern is lost. No results are retrieved from the business rules engine.
- **Assert facts, execute rule set, and retrieve results:** The same as **Assert facts and execute rule set**, except that the results are retrieved from the business rules engine. You map the results of rule set execution to BPEL variables with an assign activity. The rules session remains active. This enables you to reuse previously asserted facts.
- **Assert facts, execute rule set, retrieve results, and reset the session:** The same as **Assert facts, execute rule set, and retrieve results**, except that the results are reset for the next time that you invoke the web service. Resetting the session clears the previously asserted fact values.
- **Execute function:** Select to execute a function. Functions are also defined in dictionaries. For rule sets, you select input and output facts. For functions, you work with a fixed set of input parameters and a single return value.
- **Execute function and reset the session:** The same as **Execute function**, except that a stateful rule session is created for this pattern. All fact values are reset after retrieving the return value of the function.

To use the decision service wizard to create a decision service partner link:

1. Drag and drop a **Decision Service** under one of the **Services** swim lanes.
2. Use the Decision Service Wizard to specify the invocation pattern and rule set or function, as shown in [Figure 7–19](#).

Figure 7–19 Decision Service Wizard

Service Name:

Namespace:

Invocation Pattern: Execute function and reset the session ▼

Function:

- Assert facts only
- Retrieve results
- Assert facts and execute rule set
- Assert facts, execute rule set and retrieve results
- Assert facts, execute rule set, retrieve results and reset the session
- Execute function
- Execute function and reset the session**

- a. Enter a service name.
This will be the name of the partner link.
- b. Enter a namespace or accept the default.
- c. Select an invocation pattern.

- d. Click the flashlight icon next to the **Ruleset** or **Function** field (the field that appears depends on the invocation pattern that you selected).
 - e. Click the **Create Rule Engine Connection** icon (the icon on the left).
3. Use the Rule Explorer to create a connection to a business rules engine and select a rule set or function.

After you select a rule set or function, you are returned to the Explorer window with a connection to the repository of the rule set or function displaying in the navigation tree.

4. Use the Create Rule Engine Connection dialog to select the type of repository in which the rule sets and functions are stored in the business rules engine.

For this release, the Oracle Rules Engine repository is supported.

5. Enter the file repository directory (or browse using the folder icon) and click **Next**.

You are returned to the Rule Explorer window with a connection to the repository of the rule set or function displayed in the navigation tree.

6. Expand the dictionary and dictionary version nodes, select the rule set, and click **OK**.

7. Use the Decision Service Wizard: Select a Result or Function dialog to specify input facts and option output (watch) facts and click **Next**.

The assert facts enable you to assert a fact to the rule set or function (send factual data to the business rules engine). The watch facts enable you to return results from the rule set or function. Watch facts only appear if you selected an invocation pattern that retrieves results.

- a. Select one of the following:

Check here to assert all descendants from the top level element: Enables you to assert all the descendants of the selected rule set or function. For example, assume that a purchase order rule set contains three items. If this check box is selected, the purchase order and all three items are asserted. If this check box is not selected, only the purchase order is asserted.

Check here to add more invocation patterns: Enables you to add additional interaction patterns to the same partner link. You can only add additional rule set invocation patterns, and not function invocation patterns.

8. Use the Decision Service Wizard: Copy XSD Files dialog to review the on-screen messages to ensure that all necessary XSD schema files for this project are imported from the repository by the wizard.

The wizard attempts to identify all the schema files in the repository that must be imported into this project. Based on this attempt, this window can display the following status messages:

- If the Decision Service Wizard finds the schema files to import, the directory paths to the files display at the top of this window. No action is required on your part.
- If the Decision Service Wizard cannot find the schema files to import, the directory paths to the files displays at the top of this window. You must manually copy these files to the specified directory.

- If this XSD schema file includes or imports other schema files, ensure that those files are copied into the `bpel\rules\xsd` subdirectory of your BPEL project indicated on-screen. Ensure that you use only relative directory paths for these schema files.
9. Click **Next**.

The decision service partner link is created.

7.8.2 How to Create a Decide Activity

The decide activity enables you to create a BPEL process activity that invokes the decision service partner link you created with the Decision Service wizard. This activity also enables you to create copy operation assignments between the fact data in your rule set or function and BPEL variables.

When complete, a decide activity consisting of assign and invoke activities to the decision service partner link is created.

To create a decide activity:

1. Drag and drop a **Decide** activity into your BPEL process.
2. Enter a name, select the decision service partner link you created, and the invocation pattern (operation to perform).
3. Click **Assign Input Facts**, then click **Create** to create mappings for the input facts.

This enables you to create assignments that map BPEL input variables to automatically created BPEL variables that correspond to the input (assert) fact type.

4. If you selected an invocation pattern that retrieves results, click **Assign Output Facts**, and then click **Create** to create mappings for the output facts.

This enables you to create assignments that map automatically created BPEL variables that correspond to the output (watch) fact type.

5. Click **OK** when complete.

A decide activity consisting of assign and invoke activities to the decision service partner link is created.

7.8.3 What Happens When You Use Business Rules and a Decide Activity

A new decision service partner link for the web service is created that interfaces with the Oracle Business Rules Rules Engine. A WSDL file based on the rule set is generated.

The Decision Service Wizard provides the following:

- Dynamic processing (provides for intelligent routing, validation of policies within a process, and constraint checks)
- Integration with the ad-hoc participant type workflow (provides policy-based task assignment, various escalation policies, and load balancing of tasks)
- Integration with business activity monitoring (sends alerts based on certain policies and dynamic processing-based reasoning)

[Example 7-8](#) shows the source code the **RequiresManualApproval** decide activity in the SOA Order Booking application.

Example 7-8 Source Code for the RequiresManualApproval Decide Activity

```

<scope name="RequiresManualApproval">
  <bpelx:annotation>
    <bpelx:pattern patternName="bpelx:decide"></bpelx:pattern>
  </bpelx:annotation>
  <variables>
    <variable name="com_oracle_demos_orderbooking_Approve" element="ns27:approve"/>
    <variable name="dsIn" messageType="ns28:assertExecuteWatchStatefulMessage"/>
    <variable name="dsOut" messageType="ns28:assertExecuteWatchStatefulDecisionMessage"/>
  </variables>
  <faultHandlers>
    <catchAll>
      <switch name="Switch_2">
        <case condition="bpws:getVariableData('inputVariable','payload','/
          client:SOAOrderBookingProcessRequest/ns4:PurchaseOrder/ns4:OrderInfo/
          ns4:OrderPrice') >= 1000">
          <assign name="Assign_4">
            <copy>
              <from expression="true()"/>
              <to variable="requiresApproval"/>
            </copy>
          </assign>
        </case>
        <otherwise>
          <assign name="Assign_4">
            <copy>
              <from expression="false()"/>
              <to variable="requiresApproval"/>
            </copy>
          </assign>
        </otherwise>
      </switch>
    </catchAll>
  </faultHandlers>
  <sequence name="Sequence_16">
    <assign name="BPEL_Header">
      <copy>
        <from expression="'DecisionService'"/>
        <to variable="dsIn" part="payload"
          query="/ns28:assertExecuteWatchStateful/@name"/>
      </copy>
      <copy>
        <from expression="concat(ora:getProcessURL(), '/decisionsservices.decs')"/>
        <to variable="dsIn" part="payload"
          query="/ns28:assertExecuteWatchStateful/ns28:configURL"/>
      </copy>
      <copy>
        <from expression="ora:getProcessId()"/>
        <to variable="dsIn" part="payload"
          query="/ns28:assertExecuteWatchStateful/ns28:bpelInstance/ns13:processId"/>
      </copy>
      <copy>
        <from expression="ora:getProcessVersion()"/>
        <to variable="dsIn" part="payload"
          query="/ns28:assertExecuteWatchStateful/ns28:bpelInstance/ns13:processVersion"/>
      </copy>
    </assign>
  </sequence>

```

```
<copy>
  <from expression="ora:getDomainId()" />
  <to variable="dsIn" part="payload"
    query="/ns28:assertExecuteWatchStateful/ns28:bpelInstance/ns13:domainId"/>
</copy>
<copy>
  <from expression="ora:getProcessURL()" />
  <to variable="dsIn" part="payload"
    query="/ns28:assertExecuteWatchStateful/ns28:bpelInstance/ns13:processURL"/>
</copy>
<copy>
  <from expression="ora:getInstanceId()" />
  <to variable="dsIn" part="payload"
    query="/ns28:assertExecuteWatchStateful/ns28:bpelInstance/ns13:instanceId"/>
</copy>
<copy>
  <from expression="ora:getConversationId()" />
  <to variable="dsIn" part="payload"
    query="/ns28:assertExecuteWatchStateful/ns28:bpelInstance/ns13:conversationId"/>
</copy>
</assign>
<assign name="BPEL_Var_To_Rule_Facts">
  <copy>
    <from expression="number(bpws:getVariableData('inputVariable','payload', '/
      client:SOAOrderBookingProcessRequest/ns4:PurchaseOrder/ns4:OrderInfo/
      ns4:OrderPrice'))" />
    <to variable="com_oracle_demos_orderbooking_Approve"
      query="/ns27:approve/ns27:price"/>
  </copy>
  <copy>
    <from expression="string(bpws:getVariableData('customerServiceResponse',
      'parameters', '/ns3:findCustomerByIdResponse/ns3:return/ns3:status'))" />
    <to variable="com_oracle_demos_orderbooking_Approve"
      query="/ns27:approve/ns27:status"/>
  </copy>
</assign>
<assign name="Facts_To_Rule_Service">
  <copy>
    <from variable="com_oracle_demos_orderbooking_Approve"/>
    <to variable="dsIn" part="payload"
      query="/ns28:assertExecuteWatchStateful/ns28:assertList/ns27:approve"/>
  </copy>
</assign>
<invoke name="Invoke" partnerLink="DecisionServicePL"
  portType="ns28:IDecisionService"
  operation="assertExecuteWatchStateful" inputVariable="dsIn"
  outputVariable="dsOut"/>
<assign name="Rule_Service_To_Facts">
  <copy>
    <from variable="dsOut" part="payload"
      query="/ns28:assertExecuteWatchStatefulDecision/ns28:resultList/ns27:approve"/>
    <to variable="com_oracle_demos_orderbooking_Approve"/>
  </copy>
</assign>
```

```

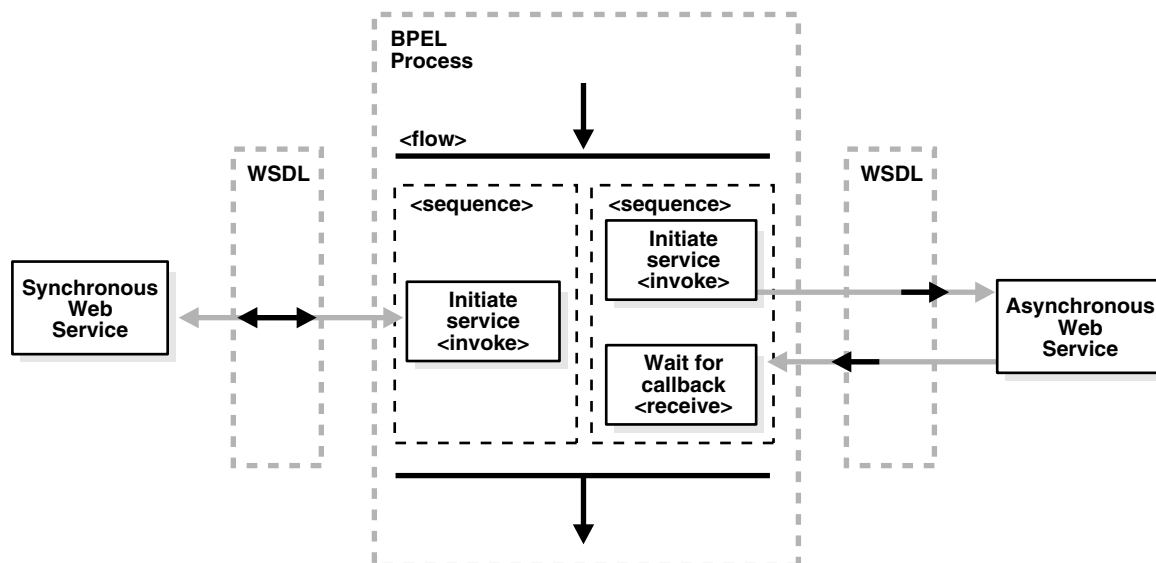
<assign name="Facts_To_BPEL_Var">
  <copy>
    <from variable="dsOut" part="payload"
      query="/ns28:assertExecuteWatchStatefulDecision/ns28:resultList/
        ns27:approve/ns27:approvalRequired"/>
    <to variable="requiresApproval"/>
  </copy>
</assign>
</sequence>
</scope>

```

7.9 Creating a Parallel Flow

Parallel flows enable a BPEL process to perform multiple tasks at the same time, which is useful when you need to perform several time-consuming and independent tasks. [Figure 7–20](#) shows a BPEL process with a parallel flow to receive a response from two different web services. The parallel flow contains two sequences, one for a synchronous web service and the other for an asynchronous web service. Both sequences use an invoke activity to initiate the service. The synchronous web service uses the invoke activity to receive the response. The asynchronous web service uses a receive activity to wait for the callback. Each response is stored in a different global variable.

Figure 7–20 Parallel Flow



7.9.1 How to Create a Parallel Flow

Create a parallel flow to send the same data to two web services. The BPEL process receives the response just like it does from any synchronous or asynchronous web service.

To create a parallel flow:

1. Within an expanded **Scope** activity, drag and drop a **Flow** activity.
2. Double-click the **Flow_1** icon.
3. Use the Flow dialog to enter a name for the flow.

The flow name does not appear when the **Flow** activity is expanded. Click the - sign to display the flow name.

7.9.2 What Happens When You Create a Parallel Flow

Using a parallel flow is more time-efficient. Without a parallel flow, each callback must call each service one at a time. By breaking the calls into a parallel flow, a BPEL process can invoke multiple web services at once, and receive the responses as they come in.

[Figure 7-21](#) shows a parallel flow, **CallManufacturers**, that is part of the SOA Order Booking application. This flow solicits price quotes from two manufacturers in parallel. The **GetSelectMfrQuote** flow invokes the **SelectService** web service, and the **GetRapidMfrQuote** invokes the **RapidService** web service. (The web services are not shown in the figure.)

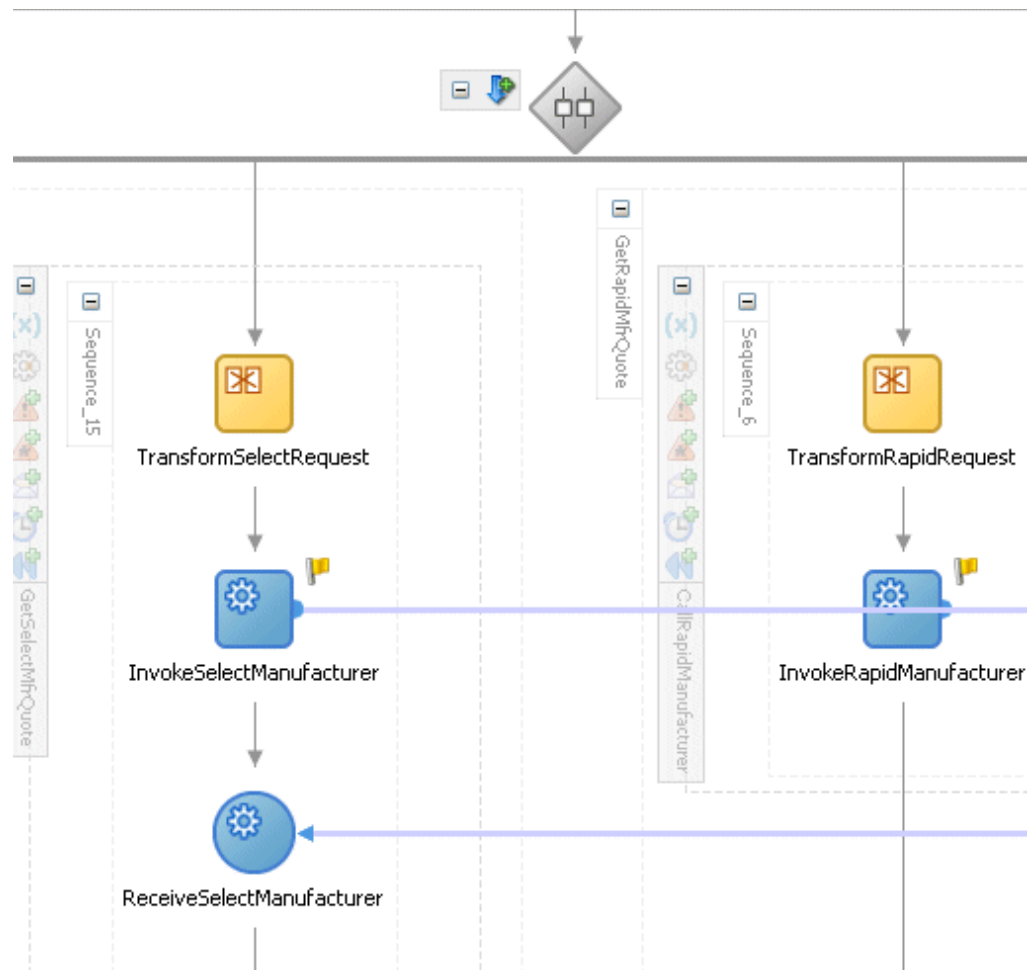
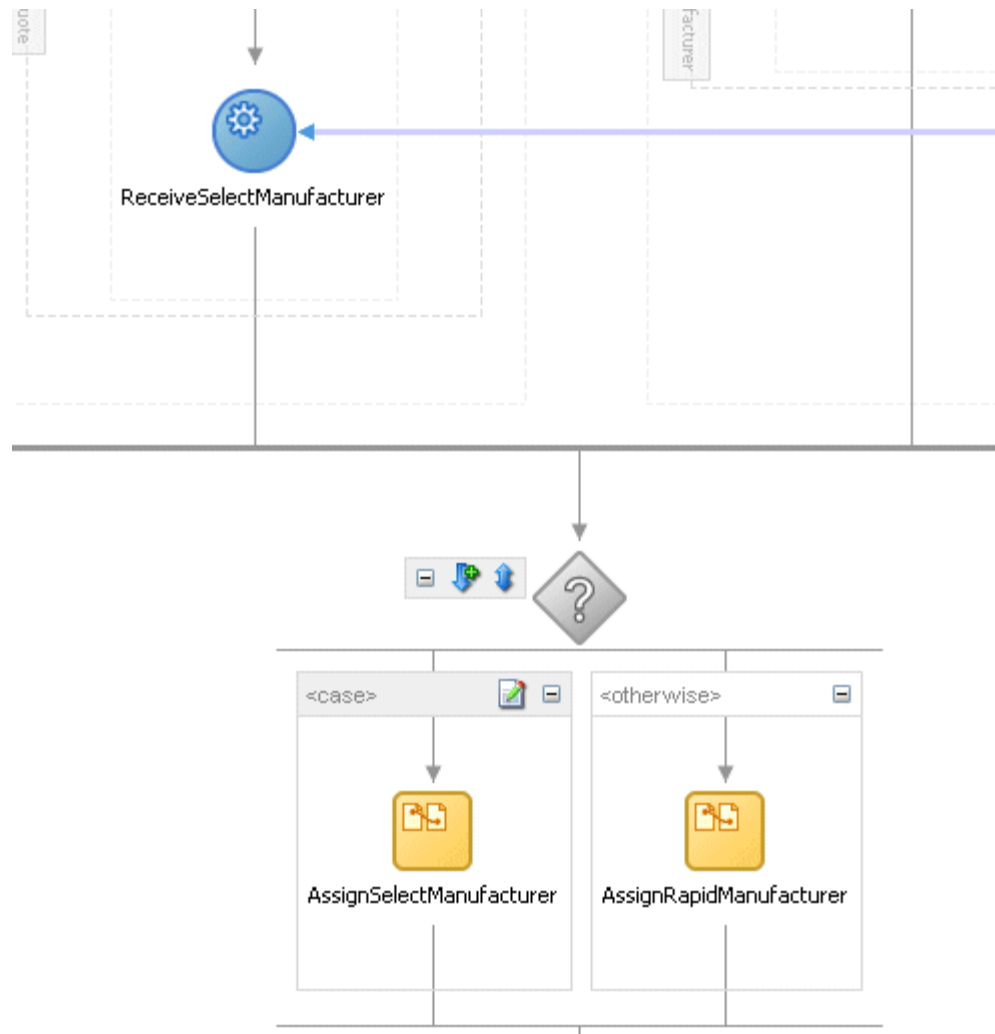
Figure 7–21 Parallel Flow in a BPEL Process

Figure 7–22 shows the continuation of the parallel flow. The two processes flow into a switch activity, where the two quotes are compared and the lower quote is selected.

Figure 7-22 *Parallel Flow Followed by a Switch Activity*

[Example 7-9](#) shows the source code for the **CallManufacturers** parallel flow in the SOA Order Booking application.

Example 7–9 Source Code for the CallManufacturers Parallel Flow

```

<scope name="SelectSupplier">
  <variables>
    <variable name="selectManufacturerResponse"
      messageType="ns17:RequestQuote_processRequestQuoteResponse" />
    <variable name="rapidManufacturerResponse"
      messageType="ns23:RequestQuotePortType_POItemsQuoteResponse" />
  </variables>
  <sequence>
    <flow name="CallManufacturers">
      <sequence name="Sequence_2">
        <scope name="CallSelectManufacturer">
          <variables>
            <variable name="manufacturerRequest"
              messageType="ns17:RequestQuote_processRequestQuote" />
          </variables>
          <sequence name="Sequence_15">
            <assign name="TransformSelectRequest">
              <bpelx:annotation>
                <bpelx:pattern>transformation</bpelx:pattern>
              </bpelx:annotation>
              <copy>
                <from expression="ora:processXSLT('SelectTransformation.xsl',
                  bpws:getVariableData('inputVariable','payload'))" />
                <to variable="manufacturerRequest" part="parameters" />
              </copy>
            </assign>
            <invoke name="InvokeSelectManufacturer"
              partnerLink="SelectService"
              portType="ns17:SelectService"
              operation="processRequestQuote"
              inputVariable="manufacturerRequest" />
            <receive name="ReceiveSelectManufacturer" createInstance="no"
              partnerLink="SelectService"
              portType="ns17:SelectServiceCallback"
              operation="processRequestQuoteResponse"
              variable="selectManufacturerResponse" />
          </sequence>
        </scope>
      </sequence>
    <sequence name="Sequence_1">
      <scope name="CallRapidManufacturer">
        <variables>
          <variable name="manufacturerRequest"
            messageType="ns23:RequestQuotePortType_POItemsQuote" />
        </variables>
        <sequence name="Sequence_6">
          <assign name="TransformRapidRequest">
            <bpelx:annotation>
              <bpelx:pattern>transformation</bpelx:pattern>
            </bpelx:annotation>
            <copy>
              <from expression="ora:processXSLT('RapidTransformation.xsl',
                bpws:getVariableData('inputVariable','payload'))" />
              <to variable="manufacturerRequest" part="parameters" />
            </copy>
          </assign>

```

```
<invoke name="InvokeRapidManufacturer"
  partnerLink="RapidService" portType="ns23:RequestQuote"
  operation="POItemsQuote"
  inputVariable="manufacturerRequest"
  outputVariable="rapidManufacturerResponse" />
</sequence>
</scope>
</sequence>
</flow>
```

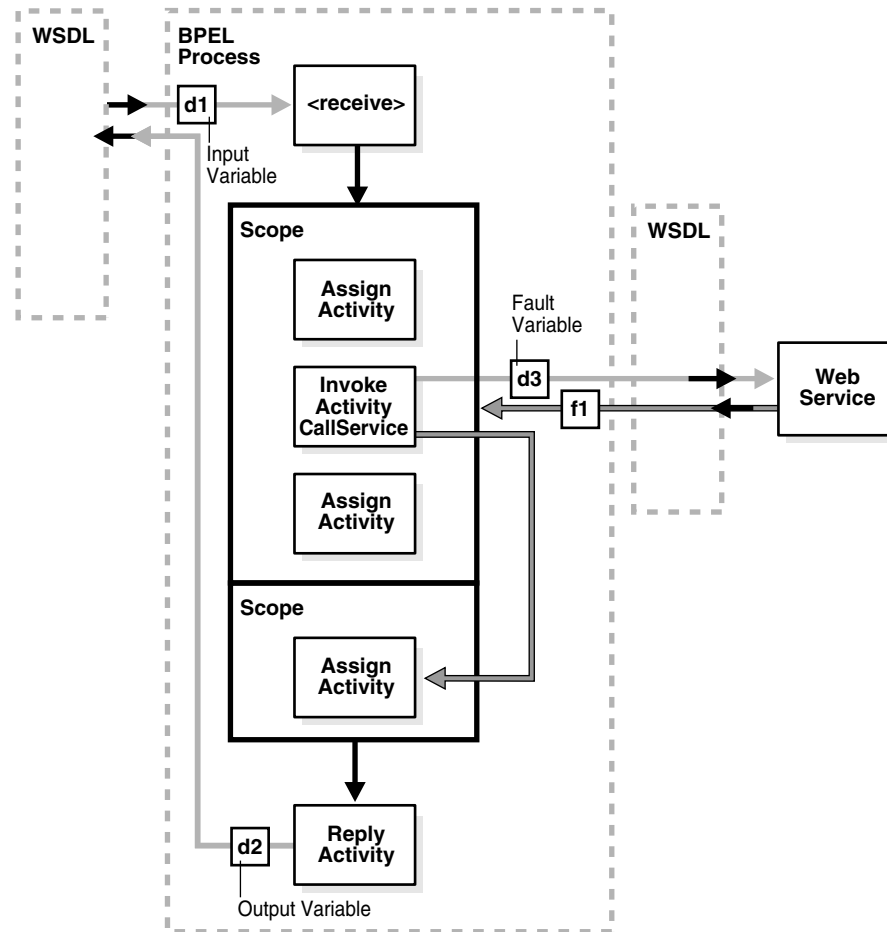
7.10 Handling Faults

Fault handling allows a BPEL process to handle error messages or other exceptions returned by outside web services, and to generate error messages in response to business or runtime faults. The following kinds of faults can occur:

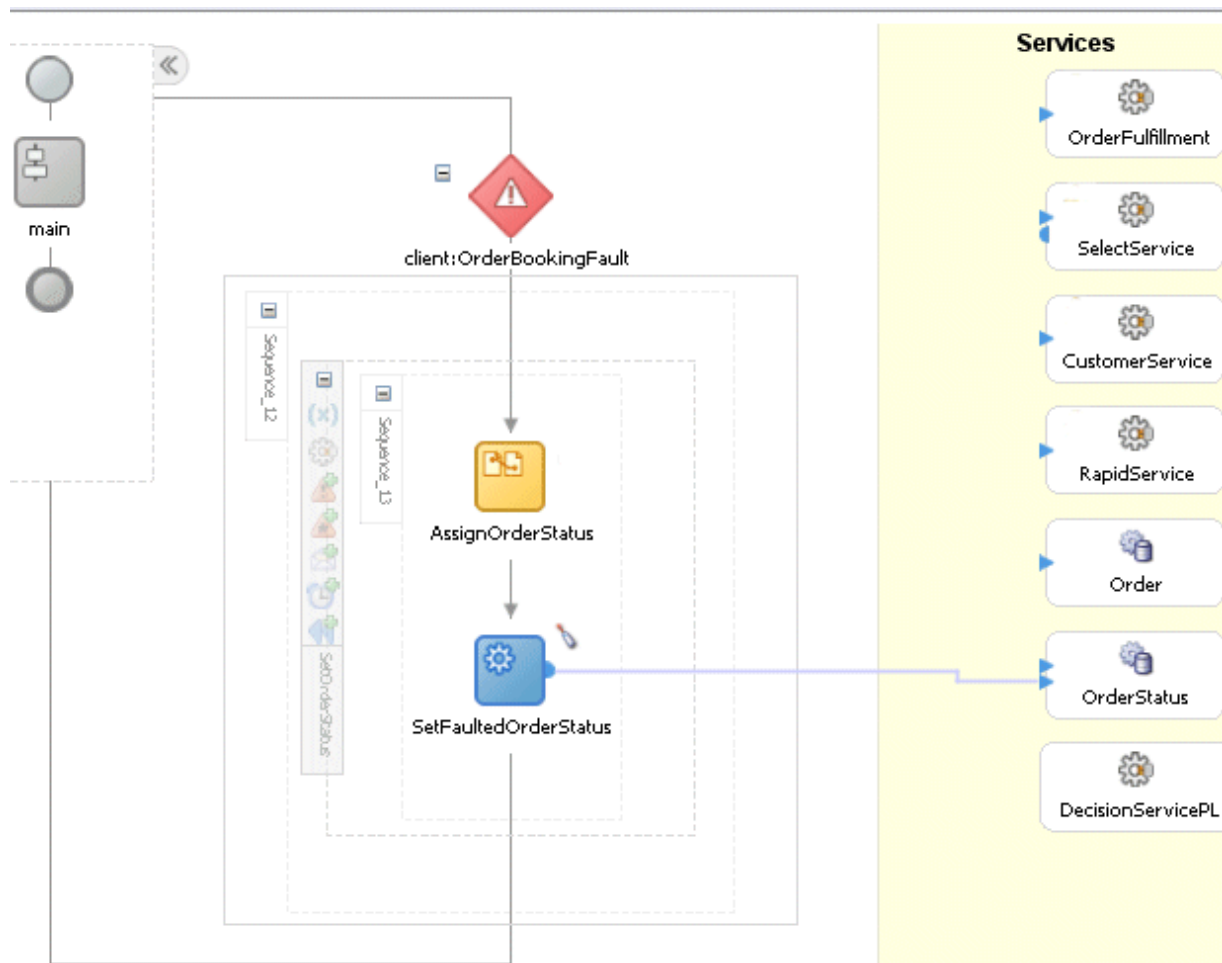
- **Business faults:** These application-specific faults are generated when an invoke activity receives a fault as a response or when an application executes a throw activity. Business faults are the result of a problem with the information, for example when the ID number of a current employee is not found in the employee database.
- **Runtime faults:** These faults are generated if a BPEL process tries to use a value incorrectly, or if there is a logic error, such as an endless loop. Runtime faults are the result of problems within the BPEL process or the web services themselves, such as when data cannot be copied properly because the variable name is incorrect.

[Figure 7–23](#) shows how fault handlers are typically incorporated within the scope activity that includes the invoke activity to a web service from which you want to handle unexpected information.

Figure 7-23 Fault Handling



In the SOA Order Booking application, the **OrderBookingFault** catchAll branch is triggered if the credit validation process fails and the order is not approved. The catchAll then sets the order status to cancelled. The **OrderBookingFault** catchAll is shown in [Figure 7-24](#).

Figure 7–24 Fault Handling with a CatchAll Branch in SOAOrderBooking.bpel

7.10.1 How to Handle Faults

Within a fault handling scope, you use one or more catch branches to catch and handle faults. Within a BPEL process, use a throw activity to generate a fault.

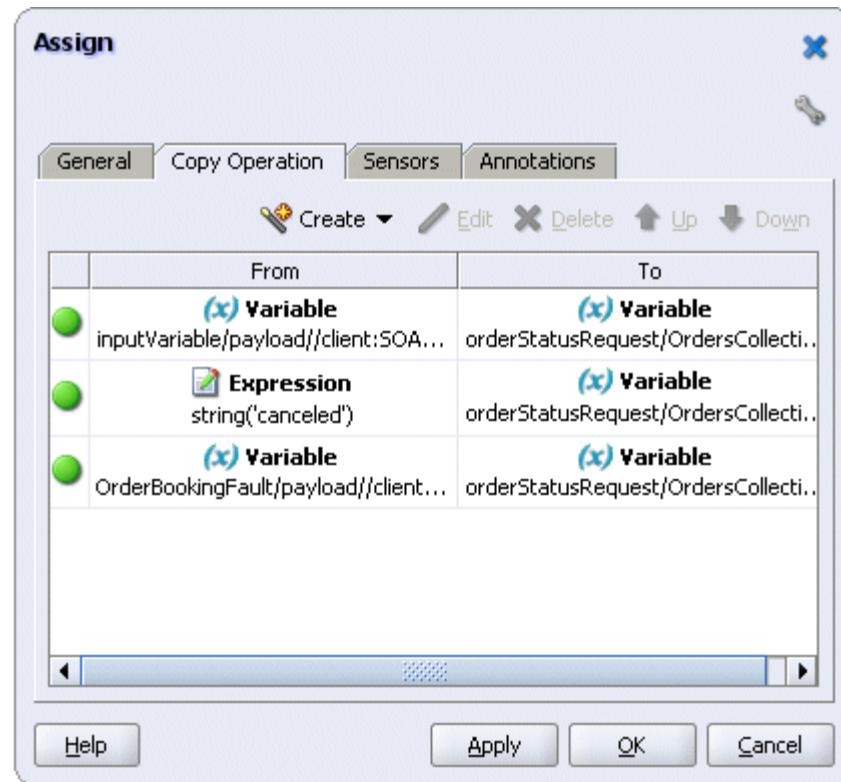
7.10.1.1 Handling Faults with a Catch or CatchAll Branch

Use a scope with a catch or catchAll branch to intercept a specific type of fault in which the data is not as expected. To catch any faults that are not already handled by name-specific catch branches, use the catchAll branch.

To handle faults with a catch or catchAll branch

1. Open the scope where you want to add a catch or catchAll branch.
2. Click the **Add Catch Branch** (or **Add CatchAll Branch**) icon.
3. Expand the branch you just created.
4. Drag and drop an **Assign** activity into the branch and double-click the **Assign** icon.
5. Use the Assign dialog to copy variables and expressions, as shown in [Figure 7–25](#).

Figure 7–25 Assign Activity as Part of a Fault Handler



6. Click **OK** to close all windows.
7. Save the file.

7.10.1.2 Handling Faults with a Throw Activity

To generate a fault from within a BPEL process, use a throw activity. When you add a throw activity to your BPEL process, it automatically includes a copy rule that copies the fault name and type into the output payload. The fault thrown by a throw activity is internal to BPEL. You cannot use a throw activity on an asynchronous process to communicate with a client. The throw activity has three elements: its name, `faultName`, and `faultVariable`. A BPEL process can send a fault to another application to indicate a problem, as opposed to throwing an internal fault. In a synchronous operation, the reply activity can return the fault. In an asynchronous operation, the invoke activity performs this function.

To handle faults with a throw activity:

1. In the BPEL process where you want to generate an error or exception message, drag and drop a throw activity.
2. Double-click the **Throw_1** icon.
3. Use the Throw dialog to enter values, as shown in [Figure 7–26](#).

Figure 7–26 *Throw Dialog*

Throw

General Sensors Annotations

Name: ThrowCreditFault

Fault QName

Namespace URI: http://www.globalcompany.com/ns/OrderBooking

Local Part: OrderBookingFault

Fault Variable: OrderBookingFault

Help Apply OK Cancel

4. Click **OK**.
5. Save the file.

7.10.2 What Happens When You Handle Faults

The `faultHandlers` tag contains the fault handling code. Within the fault handler is a catch activity, which defines the fault name and variable, and the copy instruction that sets the `creditRating` variable to `-1000`. If a fault is not handled, it creates a faulted state that migrates up through the application and can throw the entire process into a faulted state.

[Example 7–10](#) shows the source code for handling a fault using a catch branch in the SOA Order Booking application.

Example 7–10 Source Code for Fault Handling with a Catch Branch

```

<faultHandlers>
  <catch faultName="client:OrderBookingFault"
    faultVariable="OrderBookingFault">
    <sequence name="Sequence_12">
      <scope name="SetOrderStatus">
        <variables>
          <variable name="orderStatusRequest"
            messageType="ns2:OrdersCollection_msg"/>
        </variables>
        <sequence name="Sequence_13">
          <assign name="AssignOrderStatus">
            <copy>
              <from variable="inputVariable" part="payload"
                query="/client:SOAOrderBookingProcessRequest/ns4:PurchaseOrder/ns4:ID"/>
              <to variable="orderStatusRequest" part="OrdersCollection"
                query="/ns9:OrdersCollection/ns9:Orders/ns9:ordid"/>
            </copy>
            <copy>
              <from expression="string('canceled')"/>
              <to variable="orderStatusRequest" part="OrdersCollection"
                query="/ns9:OrdersCollection/ns9:Orders/ns9:status"/>
            </copy>
            <copy>
              <from variable="OrderBookingFault" part="payload"
                query="/client:SOAOrderBookingProcessFault/client:status"/>
              <to variable="orderStatusRequest" part="OrdersCollection"
                query="/ns9:OrdersCollection/ns9:Orders/ns9:comments"/>
            </copy>
          </assign>
          <invoke name="SetFaultedOrderStatus" partnerLink="OrderStatus"
            portType="ns2:OrderStatus_ptt" operation="update"
            inputVariable="orderStatusRequest"/>
        </sequence>
      </scope>
    </sequence>
  </catch>
</faultHandlers>

```

[Example 7–11](#) shows the source code for handling a fault using a throw activity in the SOA Order Booking application.

Example 7–11 Source Code for Fault Handling with a Throw Activity

```

<switch name="taskSwitch">
  <case condition="bpws:getVariableData('ApproveOrder_globalVariable', 'payload', '/task:task/
task:systemAttributes/task:state') = 'COMPLETED' and bpws:getVariableData('ApproveOrder
_globalVariable', 'payload', '/task:task/task:systemAttributes/task:outcome') = 'REJECT'">
    <bpelx:annotation>
      <bpelx:pattern>Task outcome is REJECT</bpelx:pattern>
    </bpelx:annotation>
    <sequence>
      <assign>
        <copy>
          <from expression="string('Order has been rejected by manager')"/>
          <to variable="OrderBookingFault" part="payload"
            query="/client:SOAOrderBookingProcessFault/client:status"/>
        </copy>
      </assign>
    </sequence>
  </case>
</switch>

```

```
<throw name="Throw_1" faultName="client:OrderBookingFault"
      faultVariable="OrderBookingFault"/>
</sequence>
</case>
<case condition="bpws:getVariableData('ApproveOrder_globalVariable', 'payload',
'/task:task/task:systemAttributes/task:state') = 'COMPLETED' and
bpws:getVariableData('ApproveOrder_globalVariable', 'payload', '/task:task/task:systemAttributes/
task:outcome') = 'APPROVE'">
  <bpelx:annotation>
    <bpelx:pattern>Task outcome is APPROVE</bpelx:pattern>
  </bpelx:annotation>
  <sequence>
    <empty name="Empty_1"/>
  </sequence>
</case>
<otherwise>
  <bpelx:annotation>
    <bpelx:pattern>Task is outcome is EXPIRED, STALE, WITHDRAWN or ERRORED</bpelx:pattern>
  </bpelx:annotation>
  <sequence>
    <empty name="Empty_2"/>
  </sequence>
</otherwise>
</switch>
```

7.11 Using Sensors for Monitoring

Sensors monitor BPEL activities, variables, and faults during runtime. You can define the following types of sensors, either through JDeveloper BPEL Designer or manually by providing sensor configuration files.

- Activity sensors: Used to monitor the execution of activities within a BPEL process. For example, activity sensors can be used to monitor the execution time of an invoke activity or how long it takes to complete a scope. Along with the activity sensor, you can monitor variables of the activity also.
- Variable sensors: Used to monitor variables (or parts of a variable) of a BPEL process. For example, variable sensors can be used to monitor the input and output data of a BPEL process.
- Fault sensors: Used to monitor BPEL faults.

You typically add or edit sensors as part of the BPEL modeling of activities, faults, and variables.

When you model sensors in Oracle JDeveloper BPEL Designer, two new files are created as part of the BPEL process suitcase:

- `sensor.xml`: Contains the sensor definitions of a BPEL process
- `sensorAction.xml`: Contains the sensor action definitions of a BPEL process

After you define sensors for a BPEL process, you must configure sensor actions to publish the data of the sensors to an endpoint. You can publish sensor data to the BPEL reports schema, which is located in the BPEL dehydration store, to a JMS queue or topic, or to a custom Java class.

The following information is required for a sensor action:

- Name
- Publish type

The publish type specifies the destination where the sensor data must be presented. You can configure the following publish types:

- Database: Publishes the sensor data to the reports schema in the database. The sensor data can then be queried using SQL.
- JMSQueue: Publishes the sensor data to a JMS queue
- JMSTopic: Publishes the sensor data to a JMS topic
- Custom: Publishes the data to a custom Java class
- JMS Adapter: Publishes to remote queues or topics and a variety of different JMS providers. The JMS Queue and JMS Topic publish types only publish to local JMS destinations.

- List of sensors: The sensors for a sensor action

See Chapter 18, "Sensors," in *Oracle BPEL Process Manager Developer's Guide* for more information.

7.11.1 How to Use Sensors for Monitoring

Add a sensor to a receive activity. The sensor monitors the input data received from the client.

To add a sensor to a receive activity:

1. Double-click the receive activity.
2. Click the **Sensors** tab.
3. Click the **Create** icon.
4. Use the Create Activity Sensor dialog to create the sensor.
 - a. Enter a name for the sensor.
 - b. Enter an evaluation time.

The evaluation time controls the point at which the sensor fires. You can select from the following:

Activation: The sensor fires just before the activity is executed.

Completion: The sensor fires just after the activity is executed.

Fault: The sensor fires if a fault occurs during the execution of the activity. Select this value only for sensors that monitor simple activities.

Compensation: The sensor fires when the associated scope activity is compensated. Select this value only for sensors that monitor scopes.

Retry: The sensor fires when the associated invoke activity is retried.

All: Monitoring occurs during all of the preceding phases.

- c. Click the **Create** icon to create the activity variable sensor.
- d. Use the Create Activity Variable Sensor dialog to specify the variable XPath, output namespace, and output datatype, as shown in [Figure 7-27](#).

Figure 7-27 Creating the Activity Variable Sensor

Variable XPath:

Output Namespace:

Output Datatype:

- e. Click the **Add** icon to add sensor actions.
- f. In the Sensor Action Chooser, select a sensor action to publish the values of the sensor (to a database, JMS queue, and so on).
- g. Use the Edit Sensor Action dialog to edit the sensor action you selected in the previous step, as shown in [Figure 7-28](#).

Figure 7-28 Editing the Sensor Action

Edit Sensor Action - InstanceStart

Name:

Publish Type:

JMS Connection Factory:

Publish Target:

Filter:

☒ Enable

7.11.2 What Happens When You Use Sensors

Sensor definitions are written to the `sensor.xml` file and sensor action definitions are written to the `sensorAction.xml` file, which are available from the Application Navigator, in the Sensor directory of your BPEL project.

[Example 7-12](#) shows the source code for the `sensor.xml` file in the SOA Order Booking application.

Example 7-12 Source Code for sensor.xml

```

<?xml version = '1.0' encoding = 'UTF-8'?>
<sensors targetNamespace="http://www.globalcompany.com/ns/OrderBooking"
  xmlns="http://xmlns.oracle.com/bpel/sensor"
  xmlns:tns="http://www.globalcompany.com/ns/OrderBooking"
  xmlns:pc="http://xmlns.oracle.com/bpel/sensor">
  <sensor sensorName="InstanceFaulted"
    classname="oracle.tip.pc.services.reports.dca.agents.BpelActivitySensorAgent" kind="activity"
    target="SetFaultedOrderStatus">
    <activityConfig evalTime="completion">
      <variable outputDataType="SOAOrderBookingFaultMessage"
        outputNamespace="http://www.globalcompany.com/ns/OrderBooking"
        target="$OrderBookingFault"/>
    </activityConfig>
  </sensor>
  <sensor sensorName="InstanceStart"
    classname="oracle.tip.pc.services.reports.dca.agents.BpelActivitySensorAgent" kind="activity"
    target="receiveInput">
    <activityConfig evalTime="completion">
      <variable outputDataType="SOAOrderBookingRequestMessage"
        outputNamespace="http://www.globalcompany.com/ns/OrderBooking"
        target="$inputVariable"/>
    </activityConfig>
  </sensor>
  <sensor sensorName="InstanceCompleted"
    classname="oracle.tip.pc.services.reports.dca.agents.BpelActivitySensorAgent" kind="activity"
    target="callbackClient">
    <activityConfig evalTime="completion">
      <variable outputDataType="SOAOrderBookingRequestMessage"
        outputNamespace="http://www.globalcompany.com/ns/OrderBooking" target="$inputVariable"/>
    </activityConfig>
  </sensor>
</sensors>

```

[Example 7-13](#) shows the source code for the `sensorAction.xml` file in the SOA OrderBooking application.

Example 7-13 Source Code for sensorAction.xml

```

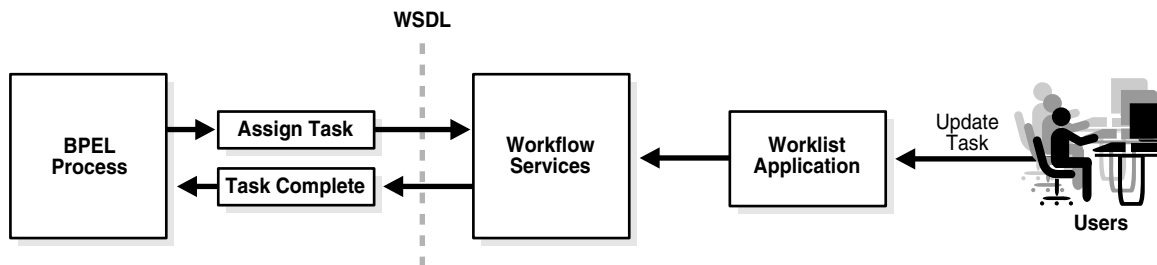
<?xml version = '1.0' encoding = 'UTF-8'?>
<actions targetNamespace="http://www.globalcompany.com/ns/OrderBooking"
  xmlns="http://xmlns.oracle.com/bpel/sensor"
  xmlns:tns="http://www.globalcompany.com/ns/OrderBooking"
  xmlns:pc="http://xmlns.oracle.com/bpel/sensor">
  <action name="InstanceStart" publishName="" publishType="JMSTopic" enabled="true"
    publishTarget="jms/demoTopic">
    <property name="JMSTopicConnectionFactory">jms/TopicConnectionFactory</property>
    <sensorName>InstanceStart</sensorName>
  </action>
  <action name="InstanceCompleted" publishName="" publishType="JMSTopic" enabled="true"
    publishTarget="jms/demoTopic">
    <property name="JMSTopicConnectionFactory">jms/TopicConnectionFactory</property>
    <sensorName>InstanceCompleted</sensorName>
  </action>
  <action name="InstanceFaulted" publishName="" publishType="JMSTopic" enabled="true"
    publishTarget="jms/demoTopic">
    <property name="JMSTopicConnectionFactory">jms/TopicConnectionFactory</property>
    <sensorName>InstanceFaulted</sensorName>
  </action>
</actions>

```

7.12 Creating a Human Workflow Task

Human workflow tasks enable you to interweave human interactions with connectivity to systems and services within an end-to-end process flow. As shown in [Figure 7–29](#), human workflow is linked to a BPEL process through a WSDL, like any other web service. The process assigns a task to a user or role and waits for a response. The users act on the task using Oracle BPEL Worklist Application (Worklist Application).

Figure 7–29 Human Workflow in Oracle BPEL Process Manager



Features of human workflow include:

- Task assignment and routing: Includes creating tasks from the business process and assigning the tasks to users or roles. Other task assignment and routing features include:
 - Support for task expiration and automatic renewal
 - Support for task delegation, escalation, and reapproval
 - Storage of task history information for auditing and the ability to archive and purge task details based on specified policies
 - JSP-based forms for viewing and updating task details
- Multiple human interaction patterns: Includes standard patterns such as simple approval, sequential approval, parallel approval, and so on. Variations on patterns such as automatic escalation, renewal, and reminders are also supported. You can also mix and match patterns to create complex patterns.
- Identity service: Interacts with back-end identity management systems to capture all user information from Java AuthoriZatioN (JAZN) and LDAP. Identity services provide role-based access control; you can assign permissions to roles and link an organizational hierarchy to a role model for authorization.
- Notification service: Notifies users of assigned tasks and task changes using various delivery channels, such as email, voice message, fax, pager, or SMS.
- The Worklist Application: Enables you to access tasks and act on them. The Worklist Application can be extended or customized based on the application.

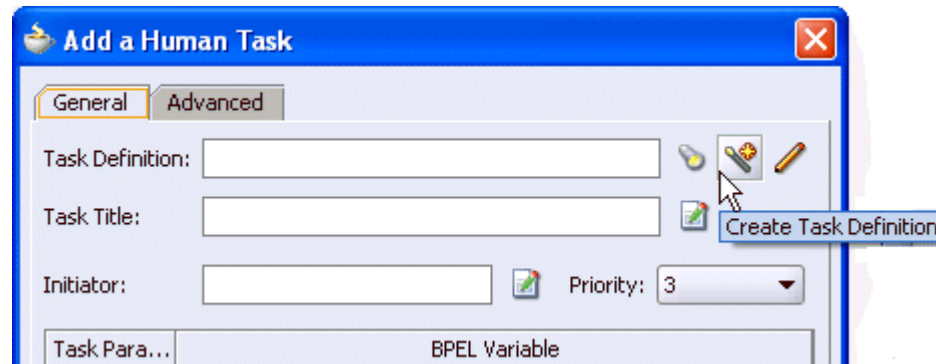
7.12.1 How to Create a Human Workflow Task

Use a human task activity and the Human Task editor to create a human workflow task. Human Task editor inputs are saved in the *human_task_name.task* file.

To create a human workflow task:

1. Drag and drop a human task activity into the BPEL process.
2. Click the **Create Task Definition** icon, shown in [Figure 7-30](#).

Figure 7-30 *Creating a Task Definition*



3. Enter a human task name and click **OK**.
4. Use the Human Task editor to configure the task.
 - a. Enter a title.
 - b. Select a priority
 - c. Use the flashlight icon to look up and select outcomes.
 - d. Enter the task owner's name, or use the flashlight icon or the expression builder icon to specify the task owner.
5. In the **Parameters** section of the Human Task editor, shown in [Figure 7-31](#), to provide parameters.

Figure 7-31 *Creating a Human Task: Parameters*



- a. Click the + icon on right side of the window.
- b. In the Add a Parameter dialog, click **Element** and then the flashlight icon.
- c. In the Type Chooser window, locate the parameter you want by expanding the **Project Schema Files** node and the XSD file node.
- d. Click **OK**.

6. In the **Assignment and Routing Policy** section of the Human Task editor, shown in [Figure 7–32](#), provide participant information.

Figure 7–32 Creating a Human Task: Assignment and Routing Policy

- a. Click the + icon on the right side of the window.
 - b. In the Add Participant Type dialog, select from **Single Approver**, **Group Vote**, **Management Chain**, **List of Single Approvers**, **FYI Assignee**, and **External Routing Service**.
 - c. Enter a label for the participant type, for example, **Approval Manager**.
 - d. Specify users or groups that must act on the task.
 - e. Specify any participants who should be skipped if the condition is satisfied.
 - f. Expand the **Advanced** section to limit the duration of the task or to allow the participant to invite other participants.
 - g. Click **OK**.
 - h. Optionally select **Allow all participants to invite other participants** or **Enable abrupt completion condition**.
7. In the **Expiration and Escalation Policy** section, shown in [Figure 7–33](#), select from **Never Expire**, **Expire after**, **Renew after**, or **Escalate after**, and specify any applicable parameters.

Figure 7–33 Creating a Human Task: Expiration and Escalation Policy

8. In the **Notification Settings** section, shown in [Figure 7–34](#), indicate how a user is assigned a task or informed that the task status has changed.

Figure 7–34 Creating a Human Task: Notification Settings

Task Status	Recipient	Notification Header
Assign	Assignees	
Complete	Initiator	
Error	Owner	

Send once, no reminders

☐ Make notifications secure (exclude details)
☐ Make email messages actionable
☐ Send task attachments with email notifications

- a. Select one of the following task status types: **Assign**, **Complete**, **Error**, **Expire**, **Request Info**, **Update Outcome**, **Suspend**, **Withdraw**.
 Use the **Assign** task status for tasks assigned to a user or a new user in a sequential list, for tasks that are renewed, delegated, reassigned, or escalated, or when information for a task is submitted.
 - b. Select one of the following recipients for the notification message: **Assignees**, **Initiator** (task creator), **Approvers**, or **Owner**.
 - c. Click the **Edit** icon to modify the default notification message text.
 This message applies to the notification channel you selected when you created a notification (email, voice, fax, pager, or SMS). Email and fax messages can also include the worklist task detail defined in this message.
9. In the **Advanced Settings** section, shown in [Figure 7–35](#), specify advanced settings, such as custom escalation rules, custom style sheets for attachments, multilingual settings, custom task actions and error messages, and callback classes.

Figure 7–35 Creating a Human Task: Advanced Settings

Specify Escalation Rule.	
Specify WordML stylesheet for attachments.	
Specify stylesheet for attachments.	
Specify multilingual settings.	Configure Resource...
Override default system actions.	Configure Actions...
Override default exception management.	Configure Assignment...
Specify callback class on task status.	Configure CallBacks...

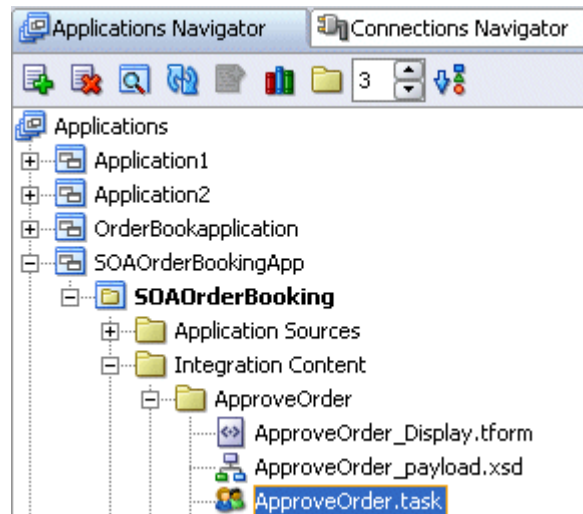
See *Oracle BPEL Process Manager Developer's Guide* for more information on the following topics:

- Sending secure notifications
- Sending actionable emails
- Sending inbound and outbound attachments
- Specifying advanced settings

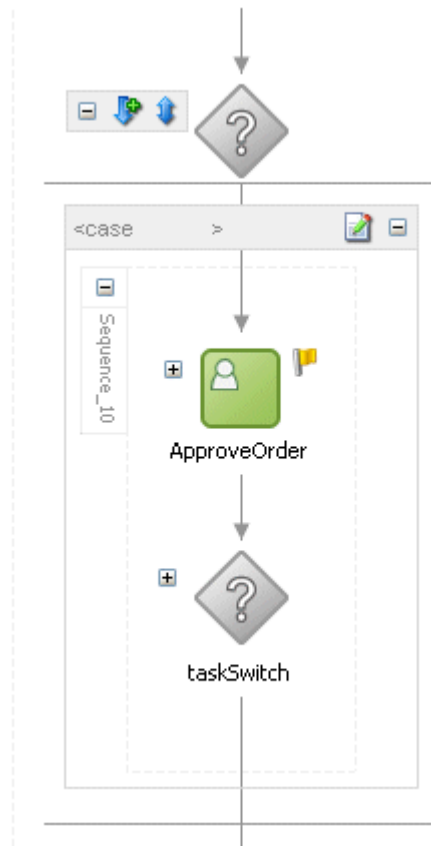
7.12.2 What Happens When You Create a Human Workflow Task

The human task settings that you specified with the Human Task editor are stored in the metadata task configuration file. This `.task` file appears in the Application Navigator, under the BPEL project Integration Content directory, as shown in [Figure 7–36](#).

Figure 7–36 The Human Task `.task` File



In the SOA Order Booking application, the `ApproveOrder.task` file is generated when you create the human workflow task **ApproveOrder**, shown in [Figure 7–37](#). The **ApproveOrder** task is part of the **requiresApproval** switch activity. The workflow enables a manager to approve or reject orders over \$1,000, for customers with Gold status.

Figure 7-37 The ApproveOrder Human Workflow

7.12.3 How to Assign Input and Output Parameters for the Human Workflow Task

Use the Expression Builder to map fields to BPEL process variables.

To assign input and output parameters:

1. Double-click the human task activity that you created previously.
2. Use the Add a Human Task dialog to complete the association of the human workflow task with the BPEL process:

- a. Enter a task title manually or use the Expression Builder to create the title dynamically.

This is a required field. Your input here overwrites the task title you entered in the Title field of the Human Task editor. This title is displayed in the Worklist Application.

- b. Enter the initiator of the task manually or use the Expression Builder.

Using the Worklist Application, the initiator can view tasks that he created and perform actions such as approving or rejecting a request. If not specified, the initiator defaults to the task owner specified on the **Advanced** tab of the Human Task dialog. The initiator defaults to `bpeladmin` if a task owner is also not specified.

- c. Select a priority value between 1 (the highest) and 5 from the **Priority** list.

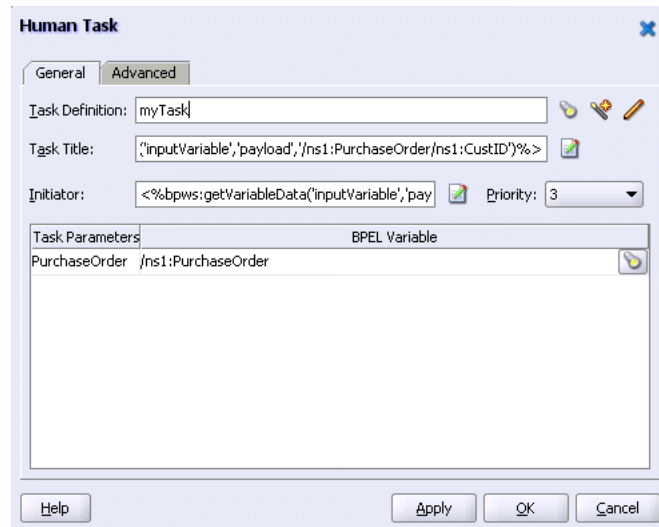
This field is provided for user reference and does not make this task a higher priority during runtime.

- d. Click the flashlight icon in the **BPEL Variable** column to map the task parameter to the BPEL variable.

You must map only the task parameters that carry input data. For output data that is filled in from the Worklist Application, you do not need to map the corresponding variables

- e. Expand the **Variables** navigation tree and select the appropriate task variable.
- f. In the Human Task dialog, shown in [Figure 7–38](#), click **Apply**.

Figure 7–38 Human Task Dialog



3. Use the **Advanced** tab of the Human Task dialog to optionally specify a scope name, global task variable name, task owner, identification key, and other advanced settings.

See *Oracle BPEL Process Manager Developer's Guide* for more information on the following advanced settings:

- Specifying a scope name and a global task variable name
- Specifying a task owner
- Specifying an identification key (a user-defined task ID)
- Including the task history of other human workflow tasks
- Allowing task and routing customizations in BPEL callbacks

7.12.4 How to Generate a Task Display Form

The task display form defines how a task payload is displayed in the Worklist Application. Task display forms are generated by using templates. Oracle JDeveloper includes three templates and two default JSPs:

The three templates are as follows:

- Three-region JSP: Consists of the header, body and footer regions. These regions can be displayed by using custom JSP or XSL or default JSP or autogenerated JSP. The autogenerated JSP displays the body region.
- Two-region JSP: Consists of the header and footer regions
- One-region JSP: Consists of the body region

The two default JSPs are as follows:

- Header JSP: Displays task attributes such as task number, priority, title, and so on
- Footer JSP: Displays task attributes such as attachment, comments, and so on

See *Oracle BPEL Process Manager Developer's Guide* for information on the following topics:

- Generating a custom task display form
- Generating a task display form with ADF Data Controls
- Creating Custom JSP forms

To generate a simple task display form:

1. Right-click the folder of the human workflow task for which you want a task display form (for example, the **ApproveOrder** task of the SOA Order Booking application).
2. Select **Auto Generate Simple Task Form**.

A simple task form is generated with the three-region template. The header region is displayed by using a `Header1.jsp` file. The footer region is displayed by using a `Footer1.jsp` file. The body region is displayed by using a `payload-body.jsp` file based on the payload.

3. Close `payload-body.jsp` by clicking the **X** in the tab.

7.12.5 What Happens When You Generate a Task Display Form

When you generate a task display form, a `.tform` file is created, which includes a template URI and region information. Workflow task display forms are deployed by using the `deployForm` ant target, which is executed when you deploy the BPEL process. The target generates an EAR file that includes all generated default or custom JSPs. The generated EAR file is deployed as a child of the Oracle BPEL Process Manager application.

[Example 7-14](#) shows the source code for the `ApproveOrder_Display.tform` file of the SOA Order Booking application.

Example 7-14 Source Code for the Task Display Form ApproveOrder_Display.tform

```

<?xml version = '1.0' encoding = 'UTF-8'?>
<taskDisplay targetNamespace="http://xmlns.companyABC.com/workflow/orderTaskDisplay"
generateInternationalizedJSP="false" xmlns:po="http://xmlns.oracle.com/bpel/workflowApproveOrder"
xmlns:task="http://xmlns.oracle.com/bpel/workflow/task"
xmlns="http://xmlns.oracle.com/bpel/workflow/taskDisplay">
  <taskDefinitionId>${domain_id}_${process_id}_${process_revision}_ApproveOrder</taskDefinitionId>
  <applicationName>worklist</applicationName>
  <templateName>ThreeRegionJSP</templateName>
  <template>${(http_url)}/${(domain_id)}/${(process_id)}/${(process_revision)}/
    ApproveOrder/ApproveOrder.jsp</template>
  <regions>
    <defaultJSP regionName="Header">
      <jspURI>Header1.jsp</jspURI>
    </defaultJSP>
    <autoGeneratedJSP regionName="Body" editable="true">
      <jspURI>payload-body.jsp</jspURI>
      <messageAttribute editable="false">PurchaseOrder</messageAttribute>
    </autoGeneratedJSP>
    <defaultJSP regionName="Footer">
      <jspURI>Footer1.jsp</jspURI>
    </defaultJSP>
  </regions>
</taskDisplay>

```

7.12.6 How to Log In to the Worklist Application

The Worklist Application uses a browser interface to enable users to act on their assigned human workflow tasks. Use Internet Explorer 6.0 or Mozilla Firefox 1.0.4 to access the Worklist Application.

See *Oracle BPEL Process Manager Developer's Guide* for more information about the Worklist Application.

To log in to the Worklist Application:

1. Open a Web browser.
2. Go to the following URL:

`http://hostname:portnumber/integration/worklistapp/Login`

- The *hostname* is the name of the host on which Oracle BPEL Process Manager is installed
- The *portnumber* used at installation (typically 8888) is noted in `bpelsetupinfo.txt`, at
SOA_Oracle_Home\install\

You can also select **Start**, then **All Programs**, then **Oracle - Oracle - soademo**, then **Oracle BPEL Process Manager**, and then **Worklist Application**.

3. Type the username and password, and click **Login**.

You can use `jcooper` and `welcome1` to log in.

7.12.7 What Happens at Runtime

When the **ApproveOrder** process runs as part of the SOA Order Booking application, the process appears in Oracle BPEL Control as an initiated activity waiting for a response, that is, waiting for a manager to approve the order. The manager uses the Worklist Application to approve or reject the request. After logging in, the manager sees a list of tasks, clicks the task title, **Approve Order**, claims the task, and then accepts or rejects it. The task details are displayed using the task display form described in [Section 7.12.4, "How to Generate a Task Display Form"](#) and shown in [Figure 7-39](#).

Figure 7-39 Worklist Application for the Approve Order Task

The screenshot shows the Oracle BPM Worklist interface. At the top, there's a header with the Oracle logo and 'BPM Worklist'. Below it, a welcome message says 'Welcome, jcooper [jzn.com]'. A navigation bar includes links for 'Home', 'Reports', 'Preferences', and 'Logout'. A status bar indicates 'Your request was processed successfully.' Below this, there are tabs for 'My Tasks' and 'Initiated Tasks'. The main content area is titled 'My Tasks > Task Details (Approve Order)'. It features a 'Task Action' dropdown menu set to '-- Select an Action --' with a 'Go' button and a 'Delegate...' button. A 'Save' button is also present. The task details are displayed in a table-like format:

[10003]	Approve Order	Created Date : 07/26/06	Creator :
State : ASSIGNED		Updated Date : 07/26/06 1:30 PM	Acquired By : jcooper
Outcome :		Expiration Date :	Assignees : Supervisor(U)
Priority : 3			

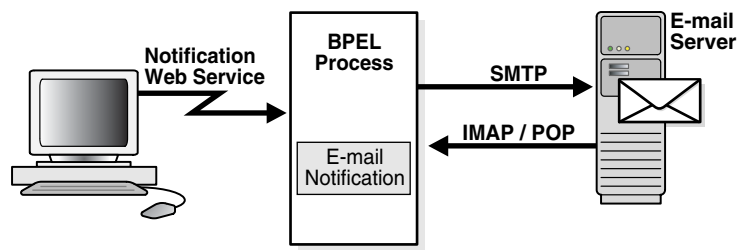
Below the task details, there are sections for 'Purchase Order' and 'Name'. The 'Purchase Order' section contains input fields for 'Cust ID' (value: 10, type: string) and 'ID' (value: 32, type: string). The 'Name' section contains input fields for 'First' (value: steve, type: string) and 'Last' (value: king, type: string).

If the manager approves the order, then the message is sent back into the BPEL flow. If the manager rejects the order, then the process cancels the order.

7.13 Creating an Email Notification

The notification service in Oracle BPEL Process Manager enables you to send notifications from a BPEL process by email, telephone voice message, fax, pager, or short message service (SMS). Notifications can be asynchronous messages to which a user does not respond, for example, a purchase order acknowledgment, or actionable messages to which a user is expected to respond, for example, an expense report approval. This section discusses email notification. See *Oracle BPEL Process Manager Developer's Guide* for information on voice, fax, pager, and SMS notification channels.

[Figure 7-40](#) shows email notification from a BPEL process. Notification services are exposed as web services. The notification service sends email using SMTP and receives email from IMAP-based or POP-based email accounts.

Figure 7-40 Email Notification from a BPEL Process

You use the email service, available from the Component Palette, to add an email notification to a BPEL process. You can directly enter the required and optional email addresses and message content (subject and body) or you can use the following built-in functions:

- Identity lookup dialog: Browse the user directory that is configured for use by Oracle BPEL Process Manager—Oracle Internet Directory (OID), JAZN/XML, LDAP, and so on. See ["To use the identity lookup dialog:"](#) on page 7-70 for instructions.
- XPath Expression Builder: For recipient email addresses (**To, Cc, Bcc, Reply To**), you can use the XPath Expression Builder to get email addresses dynamically from the input. For the subject and body of the email message, you can use the XPath Expression Builder to set dynamic text based on data from process variables that you specify. Dynamic data is automatically enclosed in `<% %>` when you use the XPath Expression Builder with free text. See ["To use the XPath Expression Builder:"](#) on page 7-71 for instructions.

See *Oracle BPEL Process Manager Developer's Guide* for information on the following topics:

- Sending email attachments
- Using Oracle Application Server Wireless in conjunction with Oracle BPEL Process Manager, which is preconfigured to send notifications using Oracle's hosted wireless service
- Starting business processes by email using the email activation agent element `activationAgents`
- Setting up an identity service

7.13.1 How to Create an Email Notification

In addition to using the email service, Identity lookup dialog, and XPath Expression Builder, you must set up configuration details for email accounts in the `ns_emails.xml` file, found at

`SOAHOME\bpel\system\services\config`

7.13.1.1 Adding an Email Service to a BPEL Process

An email activity sends an email from the BPEL process to the recipient.

To add an email activity to a BPEL process:

1. Drag and drop an **Email** service to where you want an email notification within the BPEL process.
2. Use the Edit Email dialog to enter notification details.
 - a. In the **From Account** field, enter the name of the account used to send the message.

The configuration details for this email account name must exist on Oracle BPEL Server.
 - b. In the **To**, **Cc**, **Bcc**, and **Reply To** fields, enter the email addresses to which the message is to be delivered.

This can be a static email address entered at the time the message is created, an email address looked up using the identity service, or a dynamic address from the payload.
 - c. In the **Subject** field, enter the subject of the email message.

This can be free text or dynamic text (using the XPath Expression Builder), or a combination.
 - d. In the **Body** field, enter the message body of the email message.

This can be plain text, XML, free text, or dynamic text (using the XPath Expression Builder).
 - e. Select **Multipart message with . . . attachments** and enter a number to specify email attachments.

The number includes the body. For example, if you have a body and one attachment, you enter 2.
3. Click **OK**.

7.13.1.2 Using the Identity Lookup Dialog

Use the Identity Lookup dialog to select users or groups to whom you want to send notifications by browsing the user directory (OID, JAZN/XML, LDAP, and so on) that is configured for use by Oracle BPEL Process Manager.

To use the identity lookup dialog:

1. On the Edit Email dialog, click the flashlight icon to the right of any recipient field.
2. Use the Identity lookup dialog, shown in [Figure 7-41](#), to browse the user directory (OID, JAZN/XML, LDAP, and so on) that is configured for Oracle BPEL Process Manager.

Figure 7-41 Identity Lookup Dialog Used with Email Notification

- a. Select **User** or **Group**.
- b. Select your integration server or create one.
- c. Select a realm.
For the LDAP-based JAZN provider, Oracle Internet Directory, users are seeded under the realm you selected when prompted during installation.
- d. Search by entering a search string such as `jcooper`, `j*`, `*`, and so on, and click **Lookup** to fetch all the users that match the search criterion.

- e. Highlight one or more users or groups in the search results area and click **Select**.

Use the **Hierarchy** button to view the hierarchy of a selected user. Use the **Reportees** button to display the reportees of a selected user or group. Use the **Details** button to view the details of a selected user or group.

3. Click **OK**.

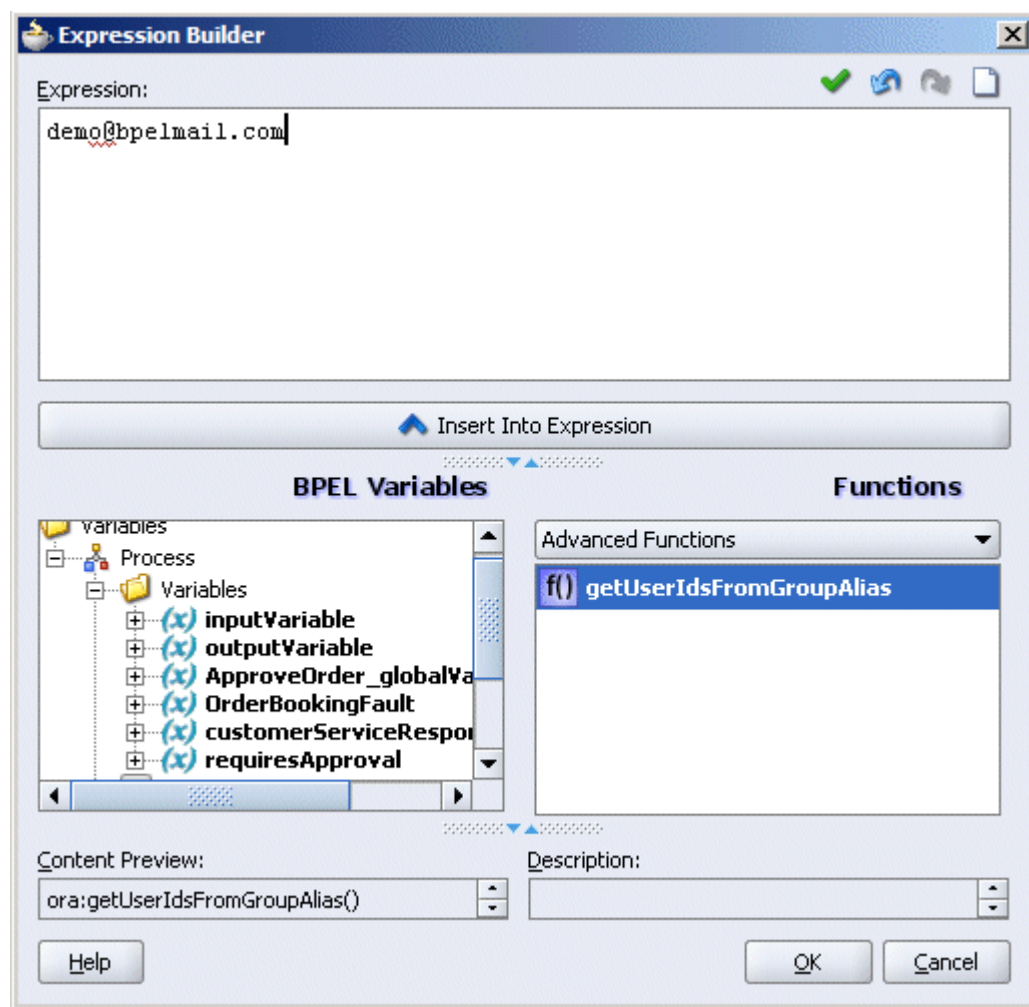
7.13.1.3 Using the XPath Expression Builder

Use the XPath Expression Builder to set up email addresses that are dynamically based on certain process variables. You can also look up contact information for a specific user using the built-in XPath functions for the identity service.

To use the XPath Expression Builder:

1. On the Edit Email dialog, click the second icon to the right of any recipient or text field.
2. Use the XPath Expression Builder, shown in [Figure 7-42](#), to get email addresses from variables or the payload.

Figure 7-42 XPath Expression Builder Used with Email Notification



Set email addresses dynamically using the following expressions:

- To get the email address (or telephone number) directly from the payload, use the following XPath:

```
bpws:getVariableData('<variable name>', '<part>', '<input xpath to get an address>')
```

For example, to get the email address from variable `inputVariable` and part payload based on XPath `/client/BPELProcessRequest/client/mail`:

```
<%bpws:getVariableData('inputVariable','payload','/client:BPELProcessRequest/client:email')%>
```

You can use the XPath Expression Builder to select the function and enter the XPath expression to get an address from the input variable.

- To get the email address or telephone number dynamically from the payload, use the following XPath:

```
ora:getUserProperty(userID, propertyName)
```

The first argument evaluates to the user ID. The second argument is the property name.

The following example gets the email address of the user identified by the variable `inputVariable`, part payload, and query `/client:BPELProcessRequest/client:userID`:

```
ora:getUserProperty(bpws:getVariableData('inputVariable',  
'payload', '/client:BPELProcessRequest/client:userid'), 'mail')
```

3. Click OK.

7.13.1.4 Configuring Email Accounts

Configure email accounts in the `ns_emails.xml` file, which is in

`SOAHOME\bpel\system\services\config`

To configure email accounts:

1. Open the `ns_emails.xml` file and configure it as follows:
 - a. Enter the sender's email address in the `FromAddress` attribute of the `EmailAccount` element.

```
<EmailAccounts EmailMimeCharset="">  
  <EmailAccount>  
    <Name>Default</Name>  
    <GeneralSettings>  
      <FromName>Oracle BPM</FromName>  
      <FromAddress>bpm1@.us.oracle.com</FromAddress>  
      . . .  
    </EmailAccount>  
  </EmailAccounts>
```

- b. Ensure that a default email account is specified in the name attribute.

This account is used when no account is specified to receive an email notification. This account is also used to send task-related notifications. A default email account must always be specified in the configuration file.

- c. Ensure that the `OutgoingServerSettings` and `IncomingServerSettings` attributes are set. For actionable notifications in a workflow, both `IncomingServerSettings` and `OutgoingServerSettings` are required.
- d. Ensure that the XML elements shown in [Table 7–1](#) are configured as needed.

Table 7–1 XML Elements for the Email Notification Configuration File

Name	Description
EmailAccount/Name	Name of the account. This can be any name, but must be unique within the BPEL server.
EmailAccount/GeneralSettings/FromName	Name of the From email address
EmailAccount/GeneralSettings/FromAddress	Email address for the From email address
EmailAccount/OutgoingServerSettings/SMTPHost	Name of the outgoing SMTP server
EmailAccount/OutgoingServerSettings/SMTPPort	Port of the outgoing SMTP server
EmailAccount/IncomingServerSettings/Server	Name of the incoming email server
EmailAccount/IncomingServerSettings/Port	Port of the incoming email server
EmailAccount/IncomingServerSettings/UserName	User ID of the email address
EmailAccount/IncomingServerSettings/Password	User password
EmailAccount/IncomingServerSettings/Password[encrypted]	Encrypted attribute of the password. It is <code>true</code> if the password is encrypted and <code>false</code> if it is not. Generally, you should set this to <code>false</code> when you first enter the password. The server automatically encrypts the password the first time it reads the configuration file and sets the attribute to <code>true</code> .
EmailAccount/IncomingServerSettings/UseSSL	Secure sockets layer (SSL) attribute. It is <code>true</code> if the incoming server requires SSL and <code>false</code> if it does not.
EmailAccount/IncomingServerSettings/Folder	Name of the folder from which to read the incoming messages
EmailAccount/IncomingServerSettings/PollingFrequency	Polling interval for reading messages from the incoming messages folder

2. Save the file.

7.13.2 What Happens When You Create an Email Notification

The notification service uses the email details you enter in the Edit Email dialog to create an assign activity with the default name **EmailParamsAssign**, as shown in Figure 7-43.

Figure 7-43 Notification Assign and Invoke Activities

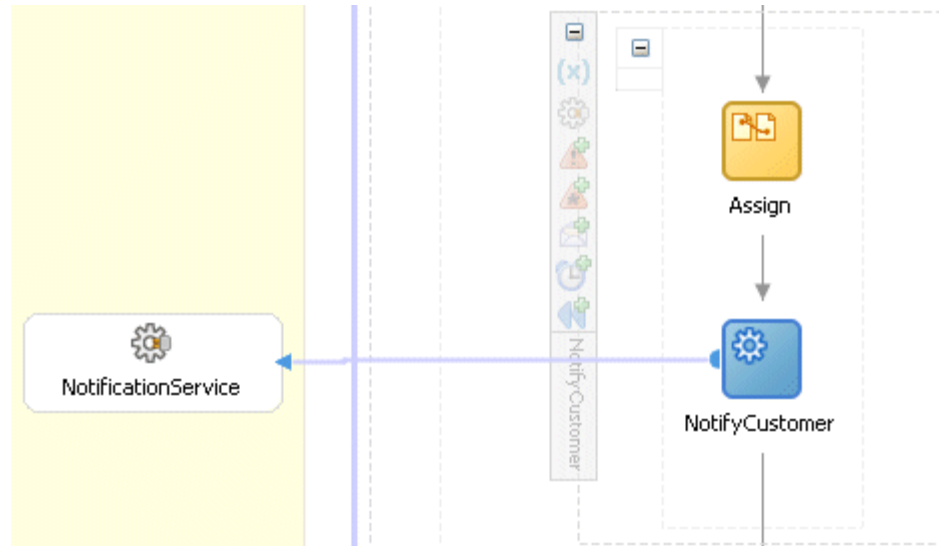
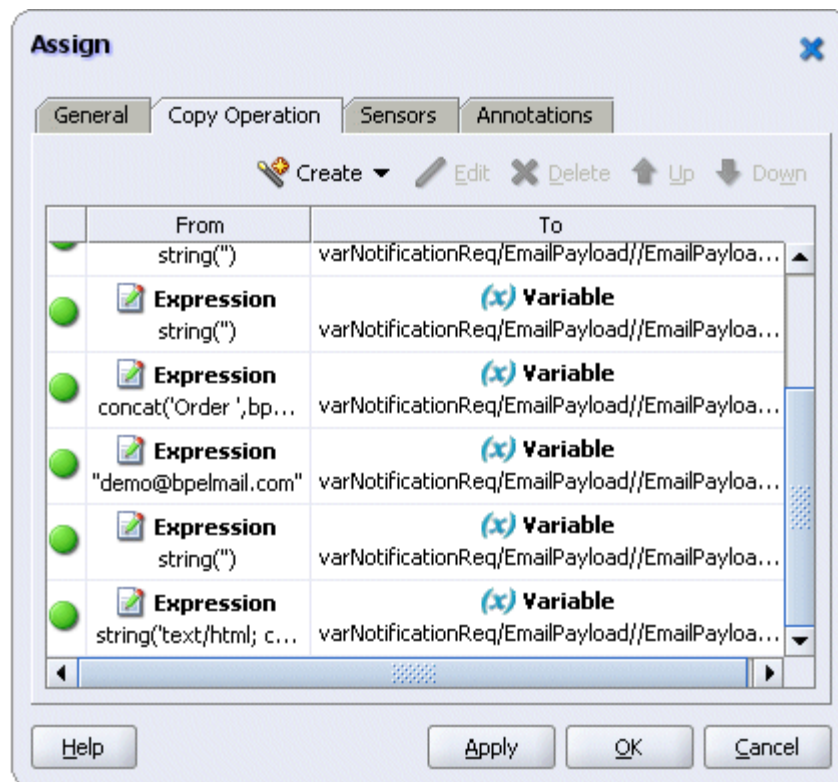
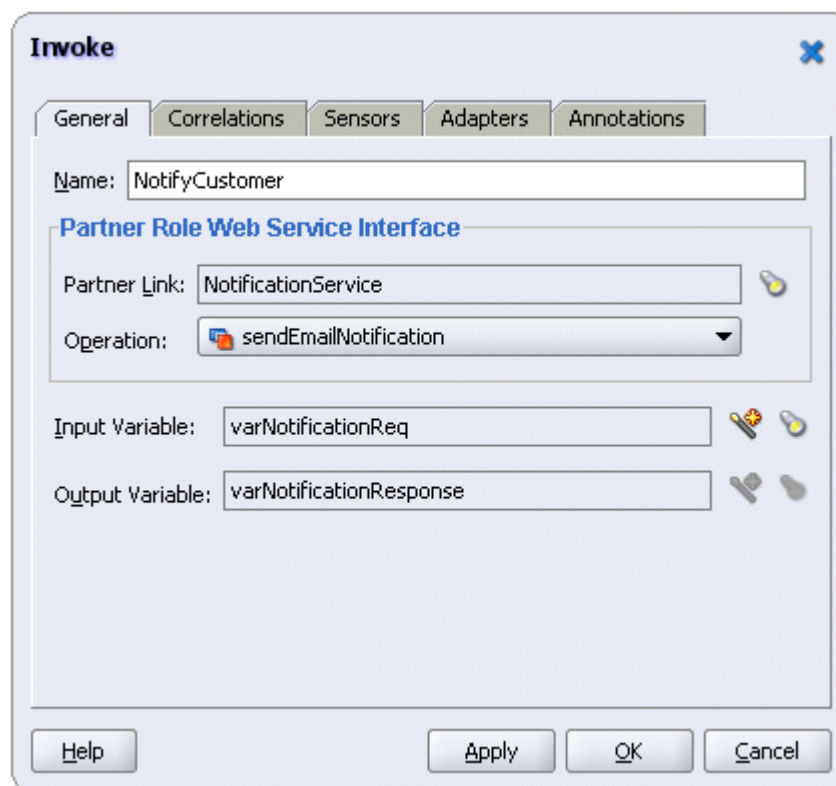


Figure 7-44 shows how the copy operations of the assign activity are automatically done for you. In the figure, `demo@bpelmail.com`, which was entered in the **To** field on the Edit Email dialog, is assigned to the variable

`varNotificationReq/EmailPayload//EmailPayload/ns5:To`

Figure 7-44 Notification Copy Operation

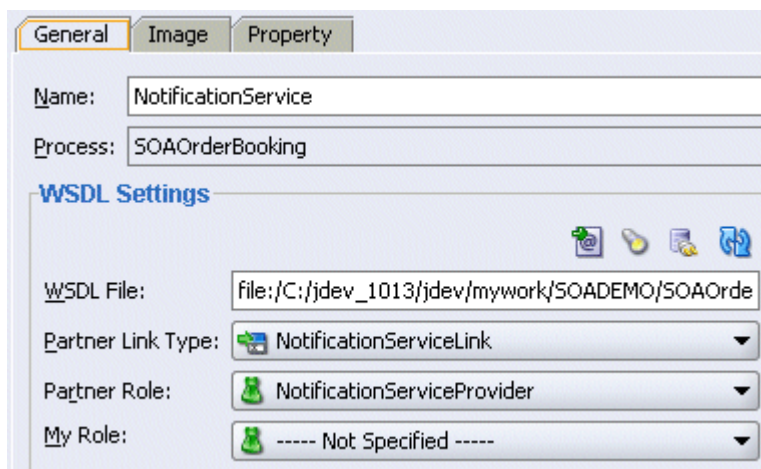
The notification service also creates an invoke activity with the default name **InvokeNotificationService** (shown in [Figure 7-43](#)) and a partner link with the default name **NotificationService**. [Figure 7-45](#) shows how the invoke activity is automatically set to interface with the **NotificationService** partner link, which is shown in [Figure 7-46](#).

Figure 7–45 Notification Invoke Dialog


The **Invoke** dialog box is shown with the **General** tab selected. It contains the following fields and controls:

- Name:** NotifyCustomer
- Partner Role Web Service Interface:**
 - Partner Link:** NotificationService
 - Operation:** sendEmailNotification
- Input Variable:** varNotificationReq
- Output Variable:** varNotificationResponse

At the bottom are buttons for **Help**, **Apply**, **OK**, and **Cancel**.

Figure 7–46 NotificationService Partner Link Settings


The **NotificationService** Partner Link Settings dialog box is shown with the **General** tab selected. It contains the following fields and controls:

- Name:** NotificationService
- Process:** SOAOrderBooking
- WSDL Settings:**
 - WSDL File:** file:/C:/jdev_1013/jdev/mywork/SOADEMO/SOAOrderBooking/NotificationService.wsdl
 - Partner Link Type:** NotificationServiceLink
 - Partner Role:** NotificationServiceProvider
 - My Role:** ----- Not Specified -----

[Example 7–15](#) shows the source code for the **NotifyCustomer** scope in the SOA Order Booking application.

Example 7-15 Source Code for the NotifyCustomer Scope for Email Notification

```

<scope name="NotifyCustomer">
  <variables>
    <variable name="varNotificationReq"
      messageType="ns5:EmailNotificationRequest"/>
    <variable name="varNotificationResponse"
      messageType="ns5:ArrayOfResponse"/>
    <variable name="NotificationServiceFaultVariable"
      messageType="ns5:NotificationServiceErrorMessage"/>
  </variables>
  <faultHandlers>
    <catch faultName="ns5:NotificationServiceFault"
      faultVariable="NotificationServiceFaultVariable">
      <empty/>
    </catch>
  </faultHandlers>
  <sequence>
    <assign name="Assign">
      <copy>
        <from expression="'Default'"/>
        <to variable="varNotificationReq" part="EmailPayload"
          query="/EmailPayload/ns5:FromAccountName"/>
      </copy>
      <copy>
        <from expression="string('')"/>
        <to variable="varNotificationReq" part="EmailPayload"
          query="/EmailPayload/ns5:Bcc"/>
      </copy>
      <copy>
        <from expression="string('')"/>
        <to variable="varNotificationReq" part="EmailPayload"
          query="/EmailPayload/ns5:Cc"/>
      </copy>
      <copy>
        <from expression="string('')"/>
        <to variable="varNotificationReq" part="EmailPayload"
          query="/EmailPayload/ns5:ReplyToAddress"/>
      </copy>
      <copy>
        <from expression="concat('Order ',bpws:getVariableData('inputVariable','payload','/
          client:SOAOrderBookingProcessRequest/ns4:PurchaseOrder/ns4:ID'),' has been completed')"/>
        <to variable="varNotificationReq" part="EmailPayload"
          query="/EmailPayload/ns5:Subject"/>
      </copy>
      <copy>
        <from expression="'demo@bpelmail.com'"/>
        <to variable="varNotificationReq" part="EmailPayload"
          query="/EmailPayload/ns5:To"/>
      </copy>
      <copy>
        <from expression="string('')"/>
        <to variable="varNotificationReq" part="EmailPayload"
          query="/EmailPayload/ns5:Content/ns5:ContentBody"/>
      </copy>
      <copy>
        <from expression="string('text/html; charset=UTF-8')"/>
        <to variable="varNotificationReq" part="EmailPayload"
          query="/EmailPayload/ns5:Content/ns5:MimeType"/>
      </copy>
    </assign>
  </sequence>
</scope>

```

```
<invoke name="NotifyCustomer" partnerLink="NotificationService"
        portType="ns5:NotificationService"
        operation="sendEmailNotification"
        inputVariable="varNotificationReq"
        outputVariable="varNotificationResponse"/>
</sequence>
</scope>
```

7.14 Compiling and Deploying a BPEL Process

After you complete the design of a BPEL process, you compile and deploy the process to Oracle BPEL Server from Oracle JDeveloper. (Oracle BPEL Server is started when the application server is started.) If compilation and deployment are successful, you can initiate and manage the BPEL process from Oracle BPEL Control. See [Chapter 13, "Monitoring Oracle BPEL Process Manager"](#) for more information.

Deployment sends the Oracle BPEL Process Manager archive (a set of files in a JAR file with a directory structure similar to the project directory structure) to Oracle BPEL Server. The deployment operation automatically validates and compiles the project directory into the BPEL archive. Therefore, you do not need to explicitly validate, compile, and recompile a project before deployment. Use Oracle BPEL Control to view any currently running BPEL processes before compiling and deploying additional processes. You must wait for the deployment of a BPEL process to complete before attempting to deploy another process.

The first time you deploy a BPEL process to Oracle BPEL Server, a default version label of 1.0 is automatically created. A version identifies a specific deployed instance of a BPEL process. The version label is appended to the end of the JAR file name created when you deploy the BPEL process.

If this label version is already deployed and the server mode is production, you are prompted to overwrite the existing version or enter a different version label. If you overwrite the version, the old process definition on the server is replaced by the new definition. You cannot revert to the old definition. In addition, any process instances that ran under the old definition are marked as stale. The stale instances cannot be examined, and all flow and audit information is lost. If you enter a different version label for the new process definition (for example, 2.0), it is deployed to Oracle BPEL Server, while the older, deployed process definition (1.0) also continues to run simultaneously on Oracle BPEL Server. The instances that ran under the old definition are retained, and not marked as stale. You can still examine the flow and audit information for these instances. If the server mode is development, you are not prompted and the version is automatically overwritten.

In a development environment, Oracle recommends that you always deploy processes to the same version on Oracle BPEL Server. This way, you avoid marking processes explicitly as the default. The life cycle to follow for this environment is as follows:

- Design your process.
- Deploy the process to Oracle BPEL Server (version is 1.0). This becomes the default process for any new instances.
- Redesign the process as needed.
- Redeploy the process as version 1.0 (this is a newer version that overwrites the older version, but version 1.0 remains the default process).

In a production environment, Oracle recommends that you increment version numbers as you deploy newer versions. For example, if OrderBooking version 1.0 is running in a production environment, then deploy the newer version of OrderBooking to version 2.0. When you have adequately tested and verified your process, mark it as the default. New instances are started using this definition. All version 1.0 instances switch seamlessly to version 2.0. This enables you to decide when a process is ready for production mode. The life cycle to follow for this scenario is as follows:

- Design your process.
- Deploy the process to Oracle BPEL Server with a different version number (for example, use version 2.0 if the older default version is 1.0).
- Test version 2.0 of the process.
- Activate version 2.0 by marking it as the default process.

WARNING: Do *not* overwrite existing versions of a process with newer versions in a production environment. This marks all existing instances of the overwritten process as stale. Stale instances cannot be examined, and all flow and audit information is lost. Instead, create a separate version as described in this section and mark the newer version as the default.

7.14.1 How to Compile and Deploy a BPEL Process

Note: Oracle BPEL Process Manager provides an automated test suite framework for creating and running repeatable tests on a BPEL process. Test cases enable you to simulate the interaction between a BPEL process and its Web service partners before deployment in a production environment. See *Oracle BPEL Process Manager Developer's Guide* for more information on testing BPEL processes before deployment.

You can deploy directly to the default domain or any other domain you have created by using an integration server connection. Domains enable you to partition and manage instances of your processes.

To compile and deploy a BPEL process:

1. In the **Application Navigator**, right-click the project name (JPR file) of the process you want to deploy.
2. Select **Deploy > my_integration_server_connection > Deploy to default domain**.
3. Click **OK**.

This compiles the BPEL process.

4. Check for errors in the Log Window at the bottom of Oracle JDeveloper. Use both the **Messages** tab and the **Apache Ant** tab to review messages.

If there are no errors, deployment was successful. If there are errors, double-click line with the error to display details about the type and location of the error.

5. Make corrections and deploy again.

See *Oracle BPEL Process Manager Developer's Guide* for more information on the following topics:

- Creating a BPEL domain
- Testing a BPEL process
- Using BPEL Process Deployer
- Compiling without deploying in Oracle JDeveloper
- Changing the Oracle BPEL Server mode
- Deploying a BPEL Suitcase to a specific domain
- Undeploying a BPEL process from a specific domain
- Understanding build and compiler command line tools for deploying a BPEL process

7.14.2 What Happens When You Compile and Deploy a BPEL Process

During compilation and deployment, the BPEL process archive and its components are compiled and packaged into a JAR file known as a BPEL suitcase. This JAR file includes the following files:

- `project_name.bpel` file implementation of the process
- `project_name.wsdl` file
- `bpel.xml` deployment descriptor file
- Any other local resources that are required, such as XML schemas, Java classes or libraries

The suitcase JAR file is deployed to

`JDev_Oracle_Home\jdev\mywork\application_name\process_name\output`

The suitcase JAR file name follows the convention of `bpel_projectname_versionnumber.jar`.

7.14.3 What Happens at Runtime

After you compile and deploy a BPEL process, use Oracle BPEL Control to see running instances. If the process requires input to complete, you can initiate the input from Oracle BPEL Control. See [Chapter 13, "Monitoring Oracle BPEL Process Manager"](#) for information.

Creating and Using Rules for a Decision Service

This chapter describes how to use Oracle Business Rules Rule Author (Rule Author) to create rules and a data model, and how to use existing facts and rule definitions with BPEL. The SOA Order Booking application is used as an example to illustrate how rules defined with the BPEL Decision Service determine whether an order requires manual approval in the credit service, based on the purchase status of the customer and the amount of the current purchase order. You use Rule Author to build the data model and the rules, after Rule Author processes the XML schema that defines the purchase order.

This chapter includes the following sections:

- [Section 8.1, "Introduction to Oracle Business Rules and Rule Author"](#)
- [Section 8.2, "Creating a Rule Author User"](#)
- [Section 8.3, "Using Rule Author"](#)
- [Section 8.4, "Creating and Saving a Dictionary with Rule Author"](#)
- [Section 8.5, "Importing an XML Schema into the Data Model with Rule Author"](#)
- [Section 8.6, "Defining a Variable in the Data Model with Rule Author"](#)
- [Section 8.7, "Creating Rules with Rule Author"](#)

For more information about using a BPEL Decision Service, see [Section 7.8, "Using Business Rules and a Decide Activity"](#).

8.1 Introduction to Oracle Business Rules and Rule Author

Oracle Business Rules enables applications to rapidly adapt to regulatory and competitive pressures and to support policy changes. Oracle Business Rules enables business analysts and other non-technical professionals to alter application logic related to business policies with a graphical user interface, without the aid of programmers. For example, in the banking industry, an account manager could set the dollar amount for which a deposit hold should be placed, in response to a changed federal regulation. In a BPEL application, you can use a Decision Service to invoke Oracle Business Rules. The resulting BPEL application supports a service-oriented architecture (SOA) paradigm by leveraging web computing standards with XML to run a rules application.

Using Oracle Business Rules with a BPEL Decision Service provides the following key benefits:

- BPEL provides orchestration of services, systems, people, and partners
- Oracle Business Rules focuses on decision-making and policies
- Oracle Business Rules enables business users to participate in business processes
- Oracle Business Rules allows for changes and new rules without redeploying a process

In a SOA application, you can use rules to determine the business policies, and the business analyst can update or change the business rules based on agile business policies. A business analyst can modify rules using Oracle BPEL Control using the process descriptor tab that provides a link to Rule Author.

Rule Author lets you work with rules from anywhere using a web browser and provides a point-and-click interface for creating new rules and editing existing rules. Also, Rule Author reduces the work required for a rule developer to create rules.

Read this chapter to understand:

- How to start and use Rule Author to create and modify business rules
- How to create a data model based on XML schema
- How to write business rules that evaluate customer data input as an XML document
- How to save business rules to a repository

8.1.1 What Are Business Rules?

Business rules are statements that describe business policies. For example, a car rental company might use the following underage driver business rule:

If the age of a driver is younger than 21, then decline to rent.

An airline might use a business rule such as the following:

If a frequent flyer account has total miles for the year that are greater than 100,000, then status is Gold.

A company that sells electronics might use a business rule such as:

If a customer has Platinum status, or an order is less than \$1,000, then approve the order without manual intervention

These examples represent individual business rules. In practice, you can use Oracle Business Rules to combine many business rules.

For the car rental example, you can name the driver age rule the Under Age rule. Traditionally, business rules such as the Under Age rule are buried in application code, and might appear in a Java application as follows:

```
public boolean checkUnderAgeRule (Driver driver) {
    boolean declineRent = false;
    int age = driver.getAge();
    if( age < 21 ) {
        declineRent = true;
    }
    return declineRent;
}
```

This code is not easy for nontechnical users to read and can be difficult to understand and modify. For example, suppose that the rental company revises its policy to "Under 19", so that all drivers under 19 cannot rent a car. In many production environments, the developer must modify the application, recompile, and then redeploy the application. Using Oracle Business Rules, this process can be simplified because a rules application supports easily changing business policies.

8.1.2 Oracle Business Rules Terminology

To use the BPEL Decision Service and Oracle Business Rules, you should first understand terminology related to business rules, including the following:

- **Facts:** data or objects on which expressions are to be evaluated. For example, facts can represent a Loan Application, a Customer Credit History, or Income Data. In this chapter, the rules operate over facts containing customer data that describes the status of a customer: Gold status, Silver status, or Platinum status, based on the amount of business for the customer in a given year.

- **Rules:** expressions on the facts:

```
If (<condition involving Facts>) then <actions>
```

For example,

```
If customer status is Gold then discount is 20%
```

- **Rule Sets:** groups of rules that are generally executed as one unit.
- **Actions:** describe an activity that is performed if a rule matches in the current knowledge base, or set of facts.
- **Data Model:** In Oracle Business Rules, facts are data objects that are asserted in the Rules Engine. Rules, such as an underage driver rule, constrain and support facts. In Oracle Business Rules, a data model specifies the types of facts or business objects that you can use to create business rules. For example, for a car rental company that needs to create a rule to match the age of a driver, the driver information represents the facts used in the rule. Using Rule Author, you can define a data model and then use the objects in the data model when you create rules.
- **Dictionary:** a collection of rule sets and their associated data model that is stored in a repository with a name and a version, or multiple versions. Rule Author stores rules in a dictionary that is saved to a repository. You can create as many dictionaries as necessary, and each dictionary can have multiple versions. A BPEL Decision Service accesses a dictionary and uses the rules in the dictionary when it invokes a Decide activity.

8.1.3 Using Existing Rules and Facts

To use existing rule definitions and facts (those not created with Rule Author and stored in a repository as described in this chapter) it is helpful to understand how the Decision Service works with rules and facts.

BPEL and a Decision Service

The Decision Service enables you to model common operations that might be performed against a set of rules and XML facts that have been written to the Oracle Rules Repository. The Decision Service simplifies the design experience by enabling you to wrap rule sets as a web service that can be invoked from a business process. The Decision Service does not provide a user interface for modifying the rules (to modify the rules, use Rule Author).

When you develop other types of rules applications, you can pass information to and from the rules engine using either Java objects or XML documents.

You can leverage Java facts with BPEL by:

1. Building Java interfaces and using the `WebServiceAssembler` to expose them to BPEL
2. Using `RuleAuthor` to point to a local `JDeveloper` classes folder (Java facts will be part of a `JDeveloper` project).

Using Rules from BPEL:

To use Rules from BPEL, you:

1. Create a Rules Repository with `JDeveloper` or the Rule Author and modeling the facts using XML.
2. Create a Decision Service Partner Link in BPEL that references the Rules Repository.

Note: If the fact model is edited in the Rule Author, the service definition is affected and the BPEL partner link must be re-synchronized. See [Section 7.8.1, "How to Use the Decision Service Wizard to Create a Decision Service Partner Link"](#)

Integrating applications with Rules

You can integrate applications with the Rules engine using the Java API to the Rules Engine (`RulesSession`):

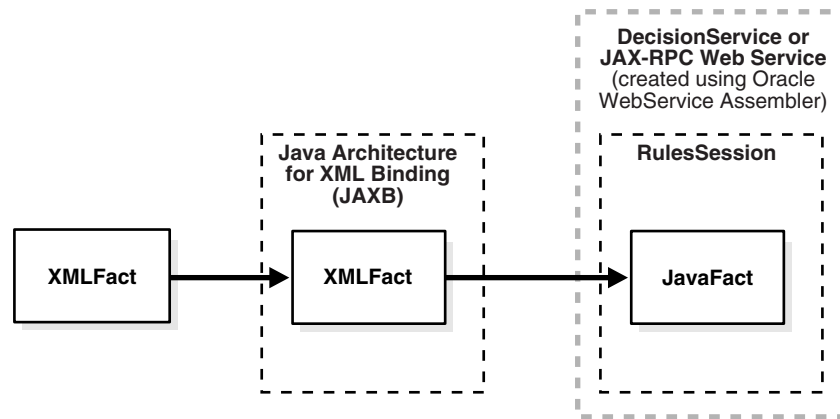
1. Build the inference engine using Java Rules, Rules RL and a runtime `RulesSession`.
2. Use `JDeveloper` to expose the inference engine as a web service, and have BPEL consume this service.

The Oracle Rules Engine implementation is itself 100% Java, and at the lowest level always operates on Java facts. To work with XML, the Rule Author uses JAXB to compile XSD into Java objects. At runtime, all Rules Engine facts are Java facts. As a developer, you can choose to:

- Model facts with XML, and have them compiled into Java seamlessly, behind the scenes, or
- Model facts using Java and expose them as a web service, where the Java-to-XML marshaling is managed by the web services stack.

In [Figure 8-1](#), `XMLFacts` are bound to `JavaFacts` using JAXB. It is always `JavaFacts` that execute in the underlying `RulesSession`.

You can interface to the `RulesSession` through the `DecisionService` (by the `XMLFact` route), or, if integration at a lower level is necessary, then you can wrap the `RulesSession/JavaFacts` with a JAX-RPC `WebService`). The preferred method is to model using `XMLFacts` and use the `DecisionService`.

Figure 8–1 Rules Integration with RulesSession or a Web Service

8.2 Creating a Rule Author User

Before you create rules with Rule Author, if you are using Oracle Application Server, you must first create a user with appropriate privileges. This allows you to log in to the Rule Author application.

8.2.1 How to Create a Rule Author User with Enterprise Manager

When your application is configured with the JAZN provider for security, you can use the default user, or you need to create a Rule Author user with Oracle Enterprise Manager 10g Application Server Control Console. If the container is not configured with JAZN XML provider, you should refer to the appropriate security documentation for information about creating users.

To create a Rule Author user:

1. In Enterprise Manager, from the Cluster Topology page, click the name of the OC4J instance where Rule Author is deployed.

Enterprise Manager displays the OC4J home page for the selected instance.

2. Click **Administration** to display a list of administration tasks you can perform on the selected OC4J instance.
3. Click the task icon in the **Security Providers** row of the table.
4. On the Security Providers page, click **Instance Level Security**.
5. On the Instance Level Security page, click **Realms**.
6. In the jazn.com row of the Results table, click the number in the Users column.

Enterprise Manager displays the list of users defined for the selected security provider.

7. Click the **Create** button.

- In the Name field, enter the name you want to use to log in to Rule Author (for example, ruleadmin).
- Enter and confirm the password for this user.
- In the Assign Roles section, double-click or use the arrows to assign the rule-administrators role to this user (see [Figure 8–2](#)).

Note: In order for Rule Author authentication to work, the user you create to log in to Rule Author must belong to the rule-administrators role.

8. When you are finished, click **OK**.
9. Return to the OC4J home page and restart the Rule Author application.

Figure 8–2 Adding a Rule Author User with Enterprise Manager

ORACLE Enterprise Manager 10g
Application Server Control

Cluster Topology > Application Server: 10g01.tvanraal-pc.us.oracle.com > OC4J: home > Security Providers > Instance Level Security > Users > Add User

Cancel OK

Realm Name: **jazzn.com**

Name: ruleadmin

Password: *****

Confirm Password: *****

Assign Roles

Available Roles

- BPMAnalyst
- BPMDDefaultDomainAdmin
- BPMSysAdmin
- BPMWorkflowAdmin
- BPMWorkflowGroupAdmin
- BPMWorkflowReassign
- BPMWorkflowSuspend
- BPMWorkflowViewHistory
- California
- CentralRegion
- EasternRegion
- LoanAgentGroup
- LoanAnalyticGroup
- RegionalOffices
- Supervisor
- WesternRegion
- ascontrol_admin
- ascontrol_appadmin
- ascontrol_monitor
- oc4j-administrators

Selected Roles

- rule-administrators

Move, Move All, Remove, Remove All

Cancel OK

8.3 Using Rule Author

Use Rule Author to create a data model describing the business objects used with rules and to create rules. You can also use Rule Author to modify an existing dictionary.

8.3.1 How to Start Rule Author

Start Rule Author by entering the URL for the home page. The URL for the home page typically includes the name of the host computer and the port number assigned to the application server during the installation, plus the path of the Rule Author home page.

To start Rule Author:

1. Enter the URL for the Rule Author home page. For example:

```
http://myhost1.mycompany.com:8888/ruleauthor/
```

Note: You can find the port number assigned to Oracle Application Server in the `readme.txt` file located in the `$ORACLE_HOME/install` directory.

2. Specify the username and password for the Rule Author user, as required by the security configuration.

The Rule Author Home page appears.

8.4 Creating and Saving a Dictionary with Rule Author

To work with Rule Author to create and modify rules, you must start with a dictionary. Rule Author stores rules and their associated definitions in a dictionary. To work with a dictionary, you can either create a new dictionary, or you can load and modify an existing dictionary.

Before you can create a dictionary, you need to set up a repository in which to store it. Rule Author supports two types of repositories for persistent storage of dictionaries, either a WebDAV (Web Distributed Authoring and Versioning) repository or a file repository. You need to determine the type of repository that you want to use to store your dictionaries. A WebDAV repository can be configured to allow users to write and access dictionaries on a web server.

8.4.1 How to Set Up a File Repository

Oracle Business Rules supplies an empty file repository (which does not contain a dictionary) with the installation. This empty file repository, named `emptyFileRepository`, is located in the `$ORACLE_HOME/rules/lib` directory. You have two choices when you need to create a new file repository: you can copy the empty file repository file to create a new repository, or you can use the Rule Author **Connect** secondary tab to create a new file repository.

To set up a new file repository using `emptyFileRepository`:

1. You can use the commands on the host system to copy the file named `emptyFileRepository` from the `$ORACLE_HOME/rules/lib` directory to the location where you want to save the file repository.
2. When you want to use the new file repository, you can connect to the repository from the Rule Author Repository Connect page (see [Section 8.4.3](#)).

To create a new file repository using Rule Author:

1. Access the Rule Author Repository Connect page.
2. Enter a full path in the **File Location** field, and then click **Create**.

After you create a repository and connect to it, you can create and save dictionaries in the repository.

8.4.2 How to Set Up a WebDAV Repository

Oracle Business Rules supports the use of a WebDAV repository in Oracle HTTP Server as the persistent storage for rule sets, the data model, and rules. Oracle HTTP Server supports WebDAV with the `mod_oradav` module.

The WebDAV protocol is an extension to the HTTP protocol that enables remote users to write content to the web server. If you use a WebDAV repository, you should configure the web server to prevent undesirable consequences and to ensure that a secure system is maintained.

It is strongly recommended that you employ some or all of these security features on the web server:

- Require authentication for access to WebDAV enabled areas
- Use of SSL, at least during authentication (for the entire session if Basic Authentication is used)
- Use of the `ForceType` directive to prevent execution for URLs that reference content in WebDAV enabled areas

To create a WebDAV repository: provides an example of how to create a WebDAV-based repository where the content is stored in the file system. This example assumes that:

- All file system paths are relative to the `ORACLE_HOME` in which the Oracle HTTP Server is installed.
- The user performing the steps is logged in to the computer as the user who installed Oracle Application Server (this ensures that the Oracle HTTP Server will be able to read from and write to `rule_repository` directory)
- Oracle HTTP Server can be accessed with the URL
`http://www.myserver.com:port`.

Note: Only use this example to configure a WebDAV repository for internal testing and not for a production environment. The repository created in this example does not include access control, and therefore allows anyone to access or modify the WebDAV repository. For information about configuring a WebDAV repository with security, see *Oracle Business Rules User's Guide*.

To create a WebDAV repository:

1. In the `Apache/Apache/htdocs` directory, create a directory named `rule_repository`.
2. In the `Apache/oradav/conf` directory, edit the `moddav.conf` file and add the following lines:

```
<Location /rule_repository>
    DAV on
    ForceType text/plain
</Location>
```

3. Restart Oracle HTTP Server.

A WebDAV repository is created, and is accessible at this URL:

```
http://www.fully_qualified_host_name.com:port/rule_
repository/
```

Note: In order for authentication to work, you must use a fully qualified host name in the URL.

For more information about using `mod_oradav`, see *Oracle HTTP Server Administrator's Guide*. In particular, see the section titled "WebDAV Security Considerations" in Chapter 9.

8.4.3 How to Create and Connect to a Repository

In Oracle Business Rules, a dictionary stores rules and the data model associated with the rules. To create and save a dictionary with Rule Author, you must first connect to a repository. Rule Author supports two types of repositories: a WebDAV repository and a file repository.

Note: If you choose to use a WebDAV repository, the repository must exist before you can connect to it.

To connect to a repository:

1. Click the **Repository** tab.
2. Click the **Connect** secondary tab.
3. Select the **File** repository type in the **Repository Type** field.
4. In the **File Location** box, enter the path to the file repository (see [Figure 8-3](#)).
5. Click **Connect**.

If you connect successfully, Rule Author shows a confirmation message.

Note: To create and connect to a file repository, click **Create** instead of **Connect**. Rule Author does not accept relative pathnames in the **File Location** field when you use **Create**.

Figure 8-3 Rule Author File Repository Connect Page

ORACLE
Rule Author

Logout Help About

Home **Repository** Definitions Rulesets Customization RL

Connect | Create | Load | Save | Save As | Properties | Import | Export | Delete

Status

Name	Value
Repository Type:	
Status	disconnected
Repository Location:	
Dictionary loaded:	none
Version:	none
Description:	

Connect

Please connect to a repository to load or create a dictionary before performing any other operations. You have to reload the dictionary if your session expires due to inactivity.

Repository Type: File

* File Location: C:\demo\sample_repository Browse...

Create Connect

8.4.4 How to Create and Save a Rule Author Dictionary

A Rule Author dictionary is the top-level container and the starting point for working with Rules. A dictionary usually corresponds to the rules portion of an application.

To create and save a dictionary:

1. Connect to a repository, as shown in [Section 8.4.3, "How to Create and Connect to a Repository"](#).
2. Click the **Create** secondary tab.
3. Enter the dictionary name in the **New Dictionary Name** field. For example, enter OrderBookingRule (see [Figure 8-4](#)).
4. Click **Create**.

Rule Author shows a status message.

Figure 8-4 Rule Author Create Dictionary Page

ORACLE
Rule Author

Logout Help About

Home Repository Definitions Rulesets Customization RL

Disconnect | Create | Load | Save | Save As | Properties | Import | Export | Delete

Status

Name	Value
Repository Type:	File
Status	connected
Repository Location:	C:\demo\sample_repository
Dictionary loaded:	none
Version:	none
Description:	

Create Dictionary
Create a dictionary in the repository.

* New Dictionary Name
Only letter, digit and underscore are allowed

Create

Existing Dictionaries

Name
(No items found)

5. Click the **Save As** secondary tab.
6. In the **Dictionary** field, enter the dictionary name.
7. To specify a version, enter a version in the **Version** field.
8. Click **Save As**.

Rule Author shows a confirmation message (see [Figure 8-5](#)).

Figure 8–5 Rule Author Save As Page

ORACLE
Rule Author

Save Dictionary Logout Help About

Home Repository Definitions Rulesets Customization RL

Disconnect Create Load Save Save As Properties Import Export Delete

Status

Name	Value
Repository Type:	File
Status	connected
Repository Location:	C:\demo\sample_repository
Dictionary loaded:	OrderBookingRule
Version:	SOASample
Description:	SOA sample order booking rule

Confirmation
Dictionary 'OrderBookingRule (SOASample)' has been saved.

Save As
Save contents to different dictionary or version

* Dictionary
Only letter, digit and underscore are allowed

* Version

Description

Overwrite ☐

8.4.5 What Happens When You Save a Dictionary

When you save a dictionary, Rule Author stores the rules and the data model associated with the dictionary in a repository using the specified version name. The save updates the repository.

8.5 Importing an XML Schema into the Data Model with Rule Author

Before defining the rules that specify conditions for a BPEL Decide Activity to evaluate using rules, you must define an Oracle Business Rules data model. A data model contains business data definitions for facts or data objects used in rules, including XML facts that are based on an XML schema.

For example, the SOA Order Booking application includes the `OrderBookingRules.xsd` file. This file specifies the schema for the customer information that the Decision Service sends as facts to the rules engine, and that the rules engine processes to determine if manual approval is required for an order. Oracle Business Rules binds an XML schema to Java classes using Java Architecture for XML Binding (JAXB). For details on using custom binding declarations, see Java Architecture for XML Binding (JAXB) Specification,

<http://java.sun.com/webservices/jaxb/index.jsp>

8.5.1 How to Import XML Schema into the Data Model

You need to process the XML schema and import the processed XML schema elements into the data model so that you can use the business objects when you define the business rules.

8.5.1.1 Performing XML Schema Processing

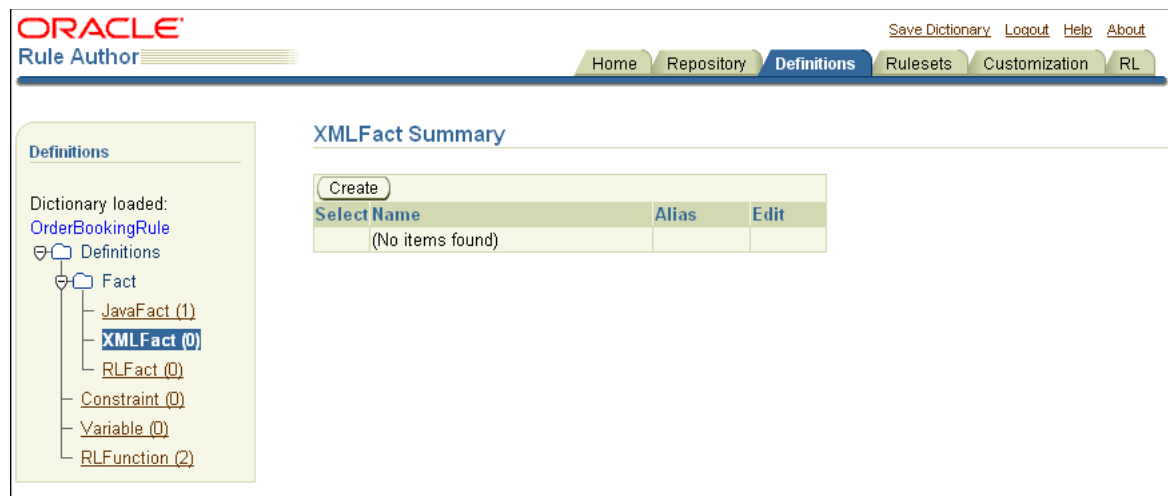
Before you can use XML elements in a data model, Rule Author must generate the classes representing the XML elements found in the XML schema. This step generates the JAXB classes and makes the generated classes and packages associated with the XML schema visible to Rule Author.

To use Rule Author to prepare a schema:

1. If necessary, load the dictionary by clicking the **Repository** tab.
2. Click the **Definitions** tab. The navigation tree shows the Definitions folder that contains the available definitions.
3. The Definitions folder in the tree contains the facts folder, which includes the available fact types: **JavaFact**, **XMLFact**, and **RLFact**. You use a **JavaFact** to work with business objects based on Java classes. A Java fact type allows selected properties and methods of a Java class to be declared so that rules can access, create, modify, and delete instances of the Java class. You use an **RLFact** to work with Oracle Business Rules RL Language fact. An RL Language fact contains a list of members of either RL Language fact type, Java fact type, or primitive type. RL Language fact types can be used to extend a business model by providing virtual dynamic types that are available when the rules engine runs.

Click **XMLFact** to view the XML Fact Summary page (see Figure 8–6). An XML fact type allows selected attributes and subelements of an XML element or complexType to be declared so that instances of it can be accessed, created, modified, and deleted by rules. For example, this is the type that you use to define the business objects in the SOA Order Booking application.

Figure 8–6 Rule Author Definitions XML Fact Summary Page



4. Click **Create**.
- Rule Author shows the XML Schema Selector page.
5. On the XML Schema Selector page, in the **XML Schema** field, enter either the path or HTTP URL for the schema. For example, you could use either of the following:
 - C:/demo/OrderBookingRules.xsd
 - http://www.myCompany.com/xsd/product.xsd
 6. In the **JAXB Class Directory** field, enter the directory where you want Rule Author to store the JAXB-generated classes (the directory that you specify must be writable). For example, enter:

c:/demo/xml

7. Enter a value for the **Target Package Name** field. If you leave this field empty, the JAXB classes package name is generated from the target namespace of the XML schema using the default JAXB XML-to-Java mapping rule. For example, the namespace `rules.oracle.com` is mapped to `com.oracle.rules`.

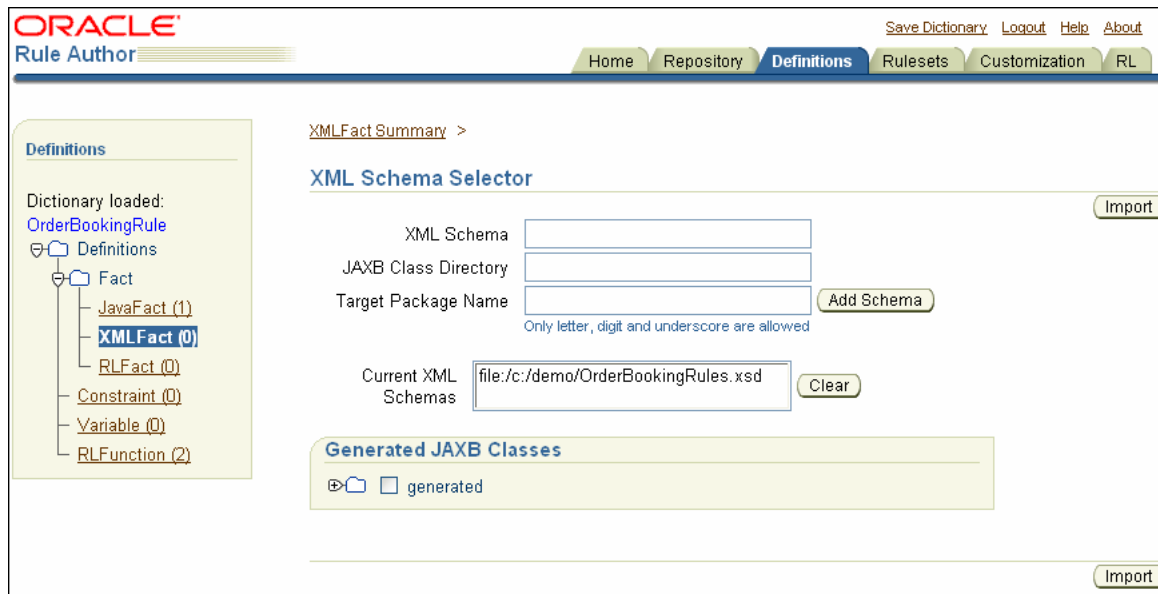
The value you enter specifies the generated classes package name. For example, generated. Although this example uses the name generated, there is nothing special about the name generated. This value specifies the package name for the JAXB-generated classes (see [Figure 8-7](#)).

Figure 8-7 Rule Author XML Schema Selection Page

The screenshot shows the Oracle Rule Author web interface. At the top, there's a navigation bar with links like 'Home', 'Repository', 'Definitions', 'Rulesets', 'Customization', and 'RL'. The 'Definitions' tab is active. On the left, a tree view shows the 'Dictionary loaded: OrderBookingRule' with a 'Definitions' folder containing 'Fact', which in turn contains 'JavaFact (1)', 'XMLFact (0)', 'RLFact (0)', 'Constraint (0)', 'Variable (0)', and 'RLFunction (2)'. The 'XMLFact (0)' item is selected. The main content area is titled 'XML Schema Selector'. It has three input fields: 'XML Schema' with the value 'c:/demo/OrderBookingRules.xsd', 'JAXB Class Directory' with 'c:/demo/xml', and 'Target Package Name' with 'generated'. Below the 'Target Package Name' field is a note: 'Only letter, digit and underscore are allowed'. There are three buttons: 'Import' (top right), 'Add Schema' (middle right), and 'Clear' (bottom left of the 'Current XML Schemas' field). Below these is a section titled 'Generated JAXB Classes' with an 'Import' button at the bottom right.

8. Click **Add Schema**.

Rule Author processes the schema and compiles the JAXB, so depending on the size of the schema, you may need to wait for this step to complete. When this step completes, the page shows the cleared **Add Schema** text entry fields and Rule Author updates the **Current XML Schemas** field and shows the **Generated JAXB Classes** box (see [Figure 8-8](#)).

Figure 8–8 Rule Author Definitions XML Schema Selector After Adding XML Schema

8.5.1.2 Importing XML Schema Elements into the Data Model

This step brings the JAXB-generated classes representing the XML schema elements into the data model (from the sample schema `OrderBookingRules.xsd`).

To import the schema representing the data required to build the rules and to output results:

1. Click the **Definitions** tab to view the Definitions page.
2. Click the **XMLFact** folder in the navigation tree.
3. Click **Create** on the XML Fact Summary page. This shows the XML Schema Selector page.
4. In the **Generated JAXB Classes** box on the XML Schema Selector page, expand the navigation tree until you see `ApproveType`, `Approve`, and `ObjectFactory`.
5. Select the generated folder check box.
6. Click **Import**.

Rule Author shows a confirmation message: "1 class or package has been imported" (see [Figure 8–9](#)).

7. Expand the Generated node in the **Generated JAXB Classes** box to see the imported classes.

Note: After an element is imported, the element is shown in bold.

Figure 8–9 Rule Author XML Schema Selector with Confirmation Message

ORACLE®
Rule Author

Save Dictionary Logout Help About

Home Repository **Definitions** Rulesets Customization RL

Definitions

Dictionary loaded:
OrderBookingRule

Definitions

- Fact
 - JavaFact (2)
 - XMLFact (3)
 - RLFact (0)
 - Constraint (0)
 - Variable (0)
 - RLFunction (2)

XMLFact Summary >

Confirmation

1 class or package has been imported.

XML Schema Selector

XML Schema

JAXB Class Directory

Target Package Name **Add Schema**

Only letter, digit and underscore are allowed

Current XML Schemas **Clear**

Generated JAXB Classes

- ☐ generated
 - ☐ ApproveType (./xs:complexType[@name='approve'])
 - ☐ Approve (./xs:element[@name='approve'])
 - ☐ ObjectFactory

Import

8.5.1.3 Saving the Dictionary

While you work on a data model from the **Definitions** tab and when you complete your work, it is important to save the dictionary.

To save a dictionary:

1. Click the **Save Dictionary** link at the top of the Rule Author page.
2. Click **Save** on the Save Dictionary page.

8.5.2 What Happens When You Import XML Schema into the Business Model

When you import XML schema into the data model, Rule Author updates the data model to include the types and elements you imported. This allows Rule Author to present these elements and types in the lists you see when you use Rule Author to create and modify rules.

When you save a dictionary, Rule Author stores the rules and the data model associated with the dictionary using the specified version name.

8.5.3 What You May Need to Know When You Import XML Schema

If you choose to access the XML schema with a URL, and a proxy server is involved, then you must set the following system properties:

```
proxyHost = $YourProxyHost
proxyPort = $YourProxyPort
proxySet = true
```

For example:

```
-DproxyHost=www-proxy.myCompany.com -DproxyPort=80 -DproxySet=true
```

For more information about setting system properties in an OC4J instance, see *Oracle Containers for J2EE Configuration and Administration Guide*.

8.6 Defining a Variable in the Data Model with Rule Author

Rule Author lets you use variables in the data model. When you use a variable, this allows you to use figurative constants in rules. For example, the data model for the SOA Order Booking application includes a variable named `AUTOMATED_ORDER_LIMIT` that specifies the business policies for the order price limit that can be approved without a manual credit approval. The SOA Order Booking application rules use a variable to define a figurative constant that defines the policies for an order value, where a value above the specified limit requires manual credit approval.

8.6.1 How to Define a Business Rules Variable in the Data Model

Use the Rule Author **Definitions** tab to add a variable to the data model. For example, in the SOA Order Booking application, we use Rule Author to add the `AUTOMATED_ORDER_LIMIT` variable to the data model.

To add a variable to the data model:

1. Click the **Definitions** tab (using the `OrderBookingRules` dictionary).
2. In the navigation tree, click the **Variable** node. This shows the Variable Summary page.
3. If you have not previously defined the `AUTOMATED_ORDER_LIMIT` variable, then click **Create**. This shows the Variable page.
4. On the Variable page, in the **Name** field, enter `AUTOMATED_ORDER_LIMIT`.
5. In the **Alias** field, enter `AUTOMATED_ORDER_LIMIT`.
6. Select the **Final** check box (by default this check box is selected).
7. In the **Type** box, select `float`.
8. In the **Expression** box, enter `1000.00`.

To use the define expression Wizard to assist you when you create an expression, click the edit icon.

9. Click **OK** (see [Figure 8–10](#)).

Note: When Rule Author creates a variable, it adds a "DM." to the name you enter in the **Name** field (DM stands for Data Model).

Figure 8–10 Rule Author Variable Definition Page

ORACLE®
Rule Author

Save Dictionary Logout Help About

Home Repository Definitions Rulesets Customization RL

Definitions

Dictionary loaded:
OrderBookingRule

Definitions

Fact

- JavaFact (2)
- XMLFact (3)
- RLFact (0)
- Constraint (0)
- Variable (1)
- RLFunction (2)

Variable Summary >

Confirmation

This entity has been updated successfully.

Variable

* Name: DM.AUTOMATED_ORDER_LIMIT

Alias: AUTOMATED_ORDER_LIMIT

Final: ☒

Type: float

Expression: 1000.00

OK Cancel Apply

8.7 Creating Rules with Rule Author

After you define the data model, you can create rules to define business policies based on the data model. For example, in the SOA Order Booking application, we define several rules that specify the business policies for an incoming order. In this rules application, the rules determine if an order requires manual credit approval.

In general, you use rules to define your business policies. For example, in the SOA Order Booking application, the rules specify that manual approval is not required if the status of the customer is platinum. For the non-platinum customers, rules specify that when the dollar amount of the order is greater than or equal to \$1000, the order requires manual approval and when the order is under \$1000, the order is approved automatically.

To define business policies in your application, you need to define rule sets and then create rules.

8.7.1 How to Create a Rule Set

Before you can create a rule, you must create a rule set. A rule set is a container for rules.

To create a rule set:

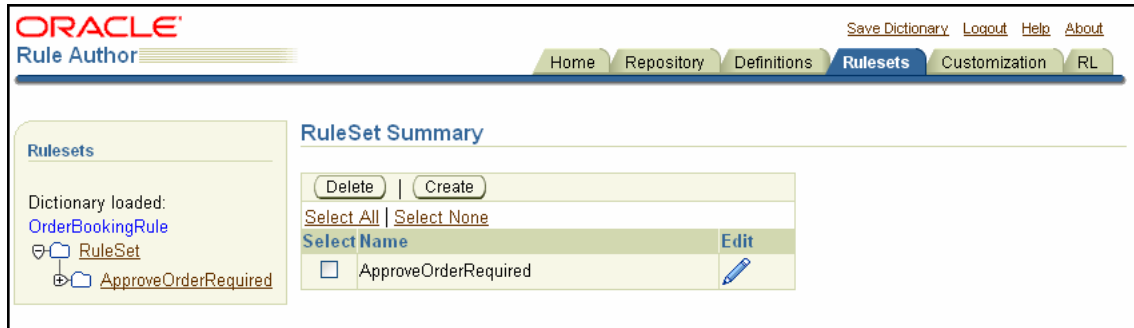
1. Click the **Rulesets** tab.
2. Click the **RuleSet** node in the navigation tree. This displays the RuleSet Summary page.
3. On the Ruleset Summary page, click **Create**. This displays the Ruleset page.
4. In the **Name** field, enter a name. For example, for the ruleset for the SOA Order Booking application, enter `ApproveOrderRequired`.
5. (Optional) Enter text in the **Description** field.

6. Click **OK**.

The rule set is created. After you create the rule set, the new entry appears in the Rulesets tree under RuleSet, as shown in [Figure 8–11](#).

7. Save the dictionary.

Figure 8–11 Rule Author Rule Set Summary Page



8.7.2 How to Create a Rule

After creating a rule set, you can create a rule within the ruleset. The rules determine your business policies. For example, in the SOA Order Booking application, the rules for the following business policies were defined:

- belowLimit: if the order is under \$1000, then the order is approved automatically.
- overLimit: for non-platinum customers, if the dollar amount of the order is greater than or equal to \$1000, then the order requires manual approval.
- platinumMember: if the customer's status is platinum, then the order is approved automatically, regardless of the dollar amount of the order.

To create a rule:

1. Click the **RuleSets** tab. The navigation pane displays the RuleSet folder. For example, for the SOA Order Booking application, the ruleset folder contains the ApproveOrderRequired rule.
2. Click a node in the ruleset folder. For example, click the **ApproveOrderRequired** node in the navigation tree. This displays the Ruleset page, with the Rules table (see [Figure 8–12](#)).

Note: If you have not yet created rules, then the Rules table is empty.

Figure 8–12 Rule Author RuleSet Page Showing the Create Button

ORACLE
Rule Author

Save Dictionary Logout Help About

Home Repository Definitions **Rulesets** Customization RL

Rulesets

Dictionary loaded:
OrderBookingRule
RuleSet
ApproveOrderRequired

RuleSet Summary >

RuleSet

* Name ApproveOrderRequired
Only letter, digit and underscore are allowed

Description

Rules

Create

Select Name

OK Cancel Apply

3. Click **Create**. This displays the Rule page.
4. On the Rule page, in the **Name** field, enter a name for the rule. For example, enter belowLimit.
5. On the Rule page in the **Priority** field, enter the rule priority for the rule. For example, enter 0 (0 is the default priority).

Note: The **Priority** field determines which rule to act upon, and in what order, if more than one rule applies. Often, in applications that use rules, the rules are applied in any order until a decision is reached, so setting a priority is not required.

6. (Optional) In the **Description** field, enter a description. For example, see Figure 8–13.

Figure 8–13 Rule Author Rule Page

ORACLE
Rule Author

Save Dictionary Logout Help About

Home Repository Definitions **Rulesets** Customization RL

Rulesets

Dictionary loaded:
OrderBookingRule
RuleSet
ApproveOrderRequired

RuleSet Summary > RuleSet >

Rule

* Name belowLimit

Description Below limit rule

Priority 0

If

New Pattern Delete

Then

New Action Delete

OK Cancel Apply

8.7.2.1 Add a Pattern for the Rule

When the rules engine runs, it uses the rules to check the available facts for matching patterns. You must add a pattern to a rule to allow the rules engine to match patterns.

To add a pattern for the rule:

1. In the **If** box on the Rule page click **New Pattern**.

The Pattern Definition page appears. The Pattern Definition page contains two areas: Choose Pattern and Define Test for Pattern.

Note: If the Pattern Definition page does not appear, you may have popup blocking enabled on your browser. Popup blocking must be disabled either just for the Rule Author application, or generally for Rule Author popup windows to appear.

2. Under **Choose Pattern**, in the first list select the first entry, which is blank.

This list specifies that the rule should fire each time there is a match. One alternative value, *There is at least one case*, selects one firing of the rule if there is at least one match (one such driver). The other alternate value, *There is no case*, specifies that the rule fires once if there are no such matches.

3. The next text box under **Choose Pattern** lets you enter a temporary name for the matched fact.

For example, enter `approve` in this field (this defines the "pattern bind variable name").

This field lets you test multiple instances of the same type in a single rule.

4. The last list contains the text, `<make a choice>`, this list shows the available fact types. In this list select `ApproveType` (see [Figure 8-14](#)).
5. Click **OK** on the pattern definition page to save the pattern definition and close the Pattern Definition page.
6. Click **OK** on the RuleSet page to save the rule.

Note: Changes made to the pattern are not added to the rule until you click **OK** or **Apply** on the Rule page. If you navigate to a different rule set or select a different tab before you click **OK** or **Apply**, Rule Author discards your pattern definition changes.

7. Save the dictionary.

Figure 8–14 Pattern Definition for Below Limit Rule

Pattern Definition OK Cancel Apply

Choose Pattern

approve is a ApproveType

Define Test for Pattern

☒ Standard Test ☐ Advanced Test

Create

	Operand	Operator	Operand (choose value or field)	
Select Field			Value	Field
	(No test)			

OK Cancel Apply

Without any tests defined on the pattern, any action that you define would apply to all approveTypes. To define tests for patterns, continue, as shown in [Section 8.7.2.2](#). For information on rule actions, see [Section 8.7.2.3, "Add an Action for a Rule"](#).

8.7.2.2 Define a Test for the Rule

Tests on patterns allow the rules engine to limit a rule match to a data item that meets the test requirements. For example, for the `belowLimit` rule, Global Company used a test to specify that the price associated with an order was below the specified approval limit.

To add a pattern test for a rule:

1. From the **Rulesets** tab, in the navigation tree click the rule where you want to add a test. For example, click the `belowLimit` rule.
2. In the **If** table on the rule page, select the pencil icon to display the Pattern Definition page for this rule.
3. On the Pattern Definition page, select **Standard Test**, then click **Create** (see [Figure 8–15](#)).

Figure 8–15 Rule Author Rule Pattern Definition Page with Standard Test Fields

Pattern Definition OK Cancel Apply

Choose Pattern

approve is a ApproveType

Define Test for Pattern

☒ Standard Test ☐ Advanced Test

Delete | Create

[Select All](#) | [Select None](#)

	Operand	Operator	Operand (choose value or field)	
Select Field			Value	Field
☐	<make a choice>	==		Fixed <make a choice> Fixed

OK Cancel Apply

4. In the **Operand** column, select a field from the list (for example, `approve.price`).
5. In the **Operator** column, select `<` (less than).
6. In the **Operand (choose value or field)** column, select a field from the list (for example, `AUTOMATED_ORDER_LIMIT`).
7. Next to the **Value** and **Field** boxes is a list containing the values Any and Fixed (see Figure 8–16). Select `Fixed` for both of these fields (this is the default).

Figure 8–16 Rule Author Pattern Definition Page with Values for the belowLimit Rule

Pattern Definition OK Cancel Apply

Choose Pattern

approve is a ApproveType

Define Test for Pattern

☒ Standard Test ☐ Advanced Test

Delete Create

[Select All](#) [Select None](#)

	Operand	Operator	Operand (choose value or field)	
			Value	Field
☐	approve.price	<	Fixed	AUTOMATED_ORDER_LIMIT Fixed

OK Cancel Apply

8. Click **OK** on the Pattern Definition page to save your changes and close the Pattern Definition page.
9. On the Rule page, click **OK** or **Apply**.

Note: Changes made to the pattern are not added to the rule until you click **OK** or **Apply** on the Rule page. If you do not click **OK** or **Apply**, Rule Author does not save your work on the rule.

10. Save the dictionary.

8.7.2.3 Add an Action for a Rule

Actions are associated with pattern matches. At runtime, when the "If" portion of a rule matches, the rules engine executes the "Then" portion to run the action or actions associated with the rule. Table 8–1 shows the Rule Author action types for actions that you can use when you create a rule.

Table 8–1 Action Types

Action Type	Description
Assert	Asserts a fact used in a pattern. If a fact matched in a pattern is changed, that fact must be asserted again to notify the rules engine that the fact has changed.
Assert New	Creates a new fact type instance and asserts that instance to the rules engine.
Assign	Assigns a value to a variable or fact property. If a new value is assigned to a fact property, the fact must be asserted again in order for the rules to be re-evaluated with the new value.
Call	Allows you to call a function to perform an action.
Retract	Retracts a fact used in a pattern. You might want to retract a fact for a number of reasons, including: ■ If you are done with the fact, and you want to remove it from the rules engine. ■ If the action associated with the rule changes the state, so that the fact must be retracted to represent the current state of the rules engine.
RL	Creates free form RL Language text that is executed directly. The syntax of the RL Language text is not validated by the SDK, so creating RL actions may result in the generation of invalid RL code from the ruleset.

To add an action that assigns values associated with a rule pattern match:

1. Click the **Rulesets** tab.
2. In the tree, click the node for the rule you want.
3. On the Rule page, in the **Then** box, click **New Action**. This displays the Add Action page.
4. From the **Action Type** box select the **Assign** item. This shows the **Action Parameters** box.
5. From the **Name** field, select the action variable you want.
6. In the Expression field, enter the expression value.
7. Click **OK** to save your changes and close the Add Action page.
8. On the Rule page, click **OK** or **Apply**.
9. Save the dictionary.

For example, to add the action for the `belowLimit` rule:

To add the action that assigns values associated with the `belowLimit` rule pattern match:

1. Click the **Rulesets** tab.
2. In the tree, under `ApproveOrderRequired`, click the `belowLimit` node.
3. On the Rule page, in the **Then** box, click **New Action**. This displays the Add Action page.
4. From the **Action Type** box select the `Assign` item. This shows the **Action Parameters** box.
5. From the **Name** field, choose `approve.approvalRequired`.
6. In the Expression field, enter the expression value `false` (see figure [Figure 8-17](#)).
7. Click **OK** to save your changes and close the Add Action page.
8. On the Rule page, click **OK** or **Apply**.
9. Save the dictionary.

Figure 8-17 Rule Author Add Action Page for the `belowLimit` Rule

8.7.3 What Happens When You Define a Ruleset and Rules

When you define a ruleset and rules, you specify business policies for your application. For example, in the SOA Order Booking application the rules specify the business policies to determine if an order requires manual credit approval based on the size of the order and the customer status. At runtime, the application runs the rules against asserted facts in the rules engine, and returns results or performs actions based on the rules that match the facts. For information on using a BPEL Decision Service, see [Section 7.8, "Using Business Rules and a Decide Activity"](#).

8.7.4 What You May Need to Know When You Define a Ruleset and Rules

When you work with BPEL and a Decision Service, you can only pass information to and from the Decision Service using an XML Document. When you develop other types of rules applications, you can pass information to and from the rules engine using either Java objects or XML documents.

Developing a Web Application in an SOA System

This chapter describes how to develop a web application that needs to work within an SOA system. Specifically it covers using the Oracle Application Development Framework (ADF) at a high level and how to use ADF to invoke web services. It also discusses how to work with web services in a web application that does not use ADF. For more detailed information about ADF, please refer to *Oracle Application Development Framework Developer's Guide*.

This chapter includes the following sections:

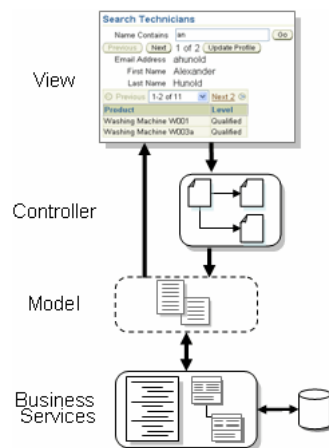
- [Section 9.1, "Introduction to Developing a Web Application in an SOA System"](#)
- [Section 9.2, "Creating a Web Application Workspace"](#)
- [Section 9.3, "Creating a Web Page"](#)
- [Section 9.4, "Creating and Using a Backing Bean for a Web Page"](#)
- [Section 9.5, "Displaying Collection and Detail Data Using ADF Data Binding"](#)
- [Section 9.6, "Collecting Data for Use by a Web Service"](#)
- [Section 9.7, "Testing the Web Application"](#)

9.1 Introduction to Developing a Web Application in an SOA System

Building web applications that will be part of an SOA system is very similar to building standalone web applications. However, it is likely that instead of communicating directly with data sources (for example, binding to business objects directly), the application will rely on web services to provide data. This chapter discusses how to create both a web application that accesses data directly, as well as one that uses a web service to access data.

Oracle ADF is an end-to-end application framework that builds on J2EE standards and open-source technologies to simplify and accelerate implementing service-oriented applications. In line with community best practices, applications you build using Oracle ADF achieve a clean separation of business logic, page navigation, and user interface by adhering to a model, view, controller architecture. As shown in [Figure 9-1](#), in an MVC architecture:

- The model layer represents the data values related to the current page
- The view layer contains the UI pages used to view/modify that data
- The controller layer processes user input and determines page navigation
- The business service layer handles data access and encapsulates business logic

Figure 9–1 MVC Architecture Cleanly Separates UI, Business Logic and Page Navigation

9.1.1 View Layer Technologies Supported

In the view layer of your application, where you design the web user interface, you can develop using either classic JavaServer Pages (JSP) or the latest JavaServer Faces (JSF) standard. You work in JDeveloper with a WYSIWYG visual designer and drag-and-drop data binding.

The SOADEMO-CLIENT application is built on the JSF standard, using the Oracle ADF Faces library. While the JSF reference implementation provides a bare-bones set of basic UI components that includes basic HTML input field types and a simple table display, the ADF Faces library works with the ADF Model layer to provide rich functionality, including:

- More sophisticated table model: Tables are a critical element of enterprise application UIs. By default, JSF doesn't support paging or sorting in tables. The ADF Faces table and the ADF Model table binding cooperate to display pageable, editable, or read-only, tables with sorting on any column.
- Key-based current selection tracking: One of the most common tasks of web user interfaces is presenting lists of information and allowing the user either to scroll through them or to select one or more entries in the list. The ADF Model iterator binding simplifies tracking the selected row in a robust way, using row keys instead of relying on positional indicators that can change when data is refreshed and positions have changed. In concert with the ADF Faces table and multiselection components, it's easy to work with single or multiple selections, and build screens that navigate master-detail information.
- More flexible models for common UI components: By default, JSF supports binding a checkbox only to boolean properties. ADF Model adds the ability to map the checkbox to any combination of true or valid values your data may present. The valid values for the list can come from any data collection in a data control and the list can perform updates or be used for row navigation, depending on your needs. The ADF Model list binding also makes null-handling easy by optionally adding a translatable "<No Selection>" choice to the list.

9.1.2 Business Services Technologies Supported by ADF Model

In the model layer, Oracle ADF Model implements the JSR-227 service abstraction called the *data control* and provides out-of-the-box data control implementations for the most common business service technologies. Whichever ones you employ,

JDeveloper and Oracle ADF work together to provide you a declarative, drag-and-drop data binding experience as you build your user interfaces. Supported technologies include:

- Enterprise JavaBeans (EJB) session beans: Since most J2EE applications require transactional services, EJB session beans are a logical choice because they offer declarative transaction control. Behind the EJB session bean facade for your business service, you use plain old Java objects (POJOs) or EJB entity beans to represent your business domain objects. JDeveloper offers integrated support for creating EJB session beans, generating initial session facade implementations, and creating either Java classes or entity beans. You can also use Oracle TopLink in JDeveloper to configure the object-relational mapping of these classes.
- JavaBeans: You can easily work with any Java-based service classes as well, including the ability to leverage Oracle TopLink mapping if needed.
- Web services: When the services your application requires expose standard web services interfaces, just supply Oracle ADF with the URL to the relevant Web Services Description Language (WSDL) for the service endpoints and begin building user interfaces that interact with them and present their results.
- XML: If your application needs to interact with XML or comma-separated values (CSV) data that is not exposed as a web service, this is easy to accomplish, too. Just supply the provider URL and optional parameters and you can begin to work with the data. This enables the incorporation of Representational State Software (REST) style services.
- ADF application modules: These service classes are a feature of the ADF Business Components module, and expose an updateable dataset of SQL query results with automatic business rules enforcement.

9.1.3 Declarative Development with Oracle ADF and JavaServer Faces

JavaServer Faces simplifies building web user interfaces by introducing web UI components that have attributes, events, and a consistent runtime API. You assemble web pages from libraries of off-the-shelf, data-aware components that adhere to the JSF standard.

In JSF you use a simple *expression language* to work with the information you want to present. Example expressions look like `#{UserList.selectedUsers}` to reference a set of selected users, `#{user.name}` to reference a particular user's name, or `#{user.role == 'manager'}` to evaluate whether a user is a manager or not. At runtime, a generic expression evaluator returns the `List`, `String`, and `boolean` value of these respective expressions, automating access to the individual objects and their properties without requiring code.

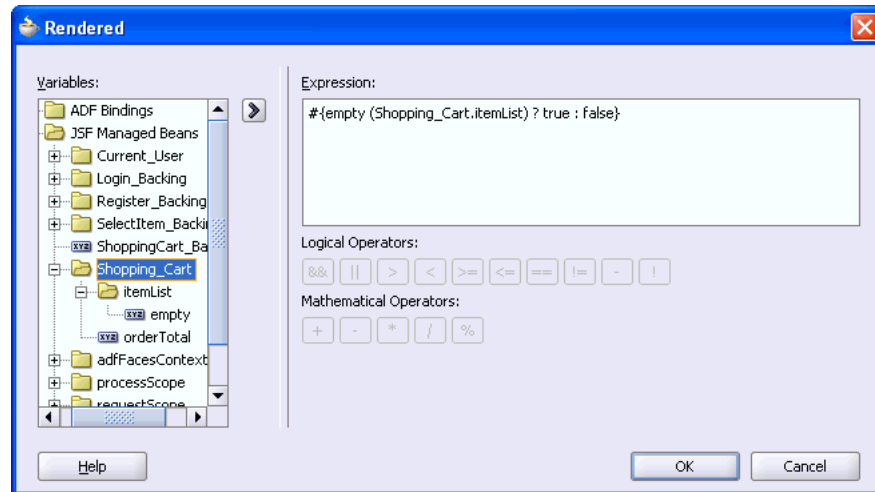
At runtime, the value of a JSF UI component is determined by its `value` attribute. While a component can have static text as its value, typically the `value` attribute will contain an EL expression that the runtime infrastructure evaluates to determine what data to display. For example, an `outputText` component that displays the name of the currently logged-in user might have its `value` attribute set to the expression `#{UserInfo.name}`.

Since any attribute of a component can be assigned a value using an EL expression, it's easy to build dynamic, data-driven user interfaces. For example, you can conditionally hide a component using a boolean-valued expression. A panel box on the cart page displays a message, but only when the cart is empty, using the expression `{empty (Shopping_Cart.itemList) ? true : false}` for the panel box's `rendered`

attribute. If the item list in the object named `Shopping_Cart` is empty, the rendered attribute evaluates to `false` and the component disappears from the page.

JDeveloper allows you to build complex expressions using a declarative tool, as shown in Figure 9-2.

Figure 9-2 The JDeveloper Expression Builder



To simplify maintenance of controller-layer application logic, JSF offers a declarative object creation mechanism. To use it, you configure the Java objects you need to use in a JSF `faces-config.xml` file. These objects are known as *managed beans* since they have properties that follow the JavaBeans specification and since the JSF runtime manages instantiating them on demand when any EL expression references them for the first time. JSF also offers a declarative mechanism to set the properties of these beans as they are first created. Managed beans can have managed properties whose runtime value is assigned by the JSF runtime based on a developer-supplied EL expression. Managed properties can depend on other beans that, in turn, also have managed properties of their own, and the JSF runtime will guarantee that the "tree" of related beans is created in the proper order.

Request-scoped managed beans that are tightly related to a given page are known colloquially as *backing beans*, since they support the page at runtime with properties and methods. The relationship between a UI component in the page and the backing bean properties and methods is established by EL expressions in a JSF page for the appropriate attributes of the component. For example:

- `value="#{expr}"`
References a property with data to display or modify
- `action="#{expr}"`
References a method to handle events
- `binding="#{expr}"`
References a property holding a corresponding instance of the UI component that you need to manipulate programmatically — show/hide, change color, and so on

Managed beans that aren't playing the role of a page's backing bean can be thought of as "application logic beans." They contain code and properties that are not specific to a single page. While not restricted to this purpose, these type of managed beans

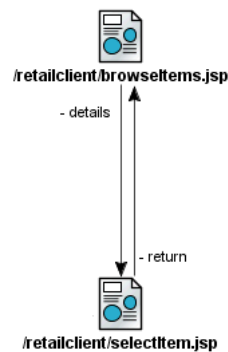
sometimes function as business service wrappers to cache method results in the controller layer beyond a single request and to centralize pre- or post-processing of business service methods that might be used from multiple pages.

In addition to using managed beans, you can also write application code in a `PhaseListener` class to augment any of the standard processing phases involved in handling a request for a JSF page. These standard steps that the JSF runtime goes through for each page are known as the "lifecycle" of the page. Most real-world JSF applications will end up customizing the lifecycle by implementing a custom phase listener of some kind, typically in order to perform tasks like preparing model data for rendering when a page initially displays.

9.1.3.1 Declarative Page Navigation with JSF

In addition to declarative UI design, JSF also provides a mechanism to declaratively define page navigation rules. Developers define these rules by specifying logical names for the legal navigation "outcomes" of a user's interaction with a page. For example, while on the `browseItems` page of the `SOADEMO-CLIENT` application, an end user may interact with the page by selecting an item and clicking the **View Details** button. The logical navigation outcome of this interaction is to go to the `selectItem` page to see more details. On the `selectItem` page, an end user might want to return to the `browseItems` page. As shown in [Figure 9-3](#), you might create names like `details` and `return` to describe the outcome. The navigation rule information is saved along with other configuration information in the `faces-config.xml` file, and at runtime JSF handles the page navigation based on these logical outcome names.

Figure 9-3 Visualizing JSF Navigation Rules in JDeveloper's Page Flow Diagram



9.1.3.2 Declarative Data Binding with Oracle ADF Model Layer

The Oracle ADF Model layer uses XML configuration files to drive generic data binding features. It implements the two concepts in JSR-227 that enable decoupling the user interface technology from the business service implementation: *data controls* and *declarative bindings*.

Data controls abstract the implementation technology of a business service by using standard metadata interfaces to describe the service's operations and data collections, including information about the properties, methods, and types involved. At design time, visual tools like JDeveloper can leverage the standard service metadata to simplify binding UI components to any data control operation or data collection. At runtime, the generic Oracle ADF Model layer reads the information describing your data controls and bindings from appropriate XML files and implements the two-way "wiring" that connects your user interface to your business service.

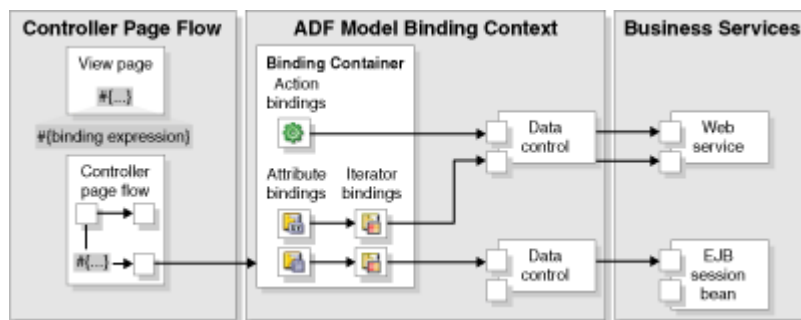
Declarative bindings abstract the details of accessing data from data collections in a data control and of invoking its operations. There are three basic kinds of declarative binding objects that automate the key aspects of data binding that all enterprise applications require:

- Iterator bindings to bind to an iterator that tracks the current row in a data collection
- Value bindings to connect UI components to attributes in a data collection
- Action bindings to invoke custom or built-in operations on a data control or its data collections

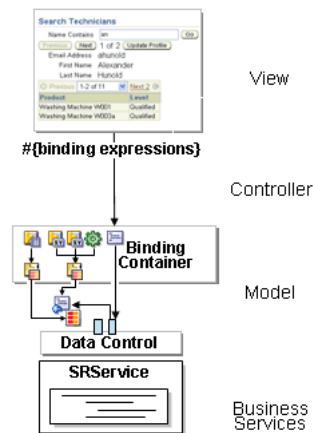
Iterator bindings simplify building user interfaces that allow scrolling and paging through collections of data and drilling-down from summary to detail information. UI components that display data use value bindings. Value bindings range from the most basic variety that work with a simple text field to more sophisticated list, table, and tree bindings that support the additional needs of list, table, and tree UI controls. An action binding is used by UI components like hyperlinks or buttons to invoke built-in or custom operations on data collections or a data control without writing code.

The group of bindings supporting the UI components on a page are described in a page-specific XML file called the *page definition file*. Generic bean factories provided by ADF Model use this file at runtime to instantiate the page's bindings. These bindings are held in a request-scoped map called the *binding container*, accessible during each page request using the EL expression `#{bindings}`. This expression always evaluates to the binding container for the current page. [Figure 9-4](#) shows how EL value binding expressions relate the UI components in a page to the binding objects in the binding container.

Figure 9-4 Bindings in the Binding Container Are EL Accessible at Runtime



[Figure 9-5](#) illustrates the architecture of a JSF application when the application also leverages the ADF Model for declarative data binding. When you combine Oracle ADF Model with JavaServer Faces, it saves you from having to write a lot of the typical managed bean code that would be required for real-world applications. In fact, many pages you build won't require a "backing bean" at all, unless you perform programmatic controller logic that can't be handled by a built-in action or service method invocation (which ADF Model can do without code for you). You can also do away with any application logic beans that wrap your business service, since the ADF Model's data control implements this functionality for you. And finally, you can often avoid the need to write any custom JSF phase listeners because ADF Model offers a generic JSF phase listener that performs most of the common operations you need in a declarative way based on information in your page definition metadata.

Figure 9–5 Architecture of a JSF Application Using ADF Model Data Binding

The core module in the framework is Oracle ADF Model, a declarative data binding facility that implements the JSR-227 specification. The Oracle ADF Model layer enables a unified approach to bind any user interface to any business service with no code.

9.1.4 Building a Web Application

The typical development process of building an Oracle ADF application starts with creating the services that will supply the data. For example, the SOADEMO-CLIENT application uses Java Persistence API (JPA) entities to map to the products in the PRODUCT database table. It also uses an EJB session bean to create the service used to aggregate and present the data. Once these are created, you can generate data controls that you can then use to create the web client. For example, the `ProductsFacadeLocal` data control in the SOADEMO-CLIENT application allows developers to drag and drop results of standard CRUD methods run on the session bean as various ADF Faces components. For more information about creating a data control for EJBs, see [Section 4.7, "Exposing Services with ADF Data Controls"](#).

There may also be instances when the web client accesses data through a web service, as the SOADEMO-CLIENT does when using the `CustomerService` web service to create a customer. You can also create data controls from the web service, or in some cases, you may need to work with the web service directly, without using the ADF Model layer. For more information about creating data controls for web services, see [Section 5.3.3, "How to Create a Data Control From a Web Service"](#). For more information about when you may need to work directly with a web service, see [Section 9.6.3, "How to Invoke a Service Using a Backing Bean"](#).

Once data controls are created, you create the pages and the navigation between them using the JSF navigation diagram. For more information, see [Section 9.3, "Creating a Web Page"](#). Once the navigation is set, you can begin to add complex capabilities to your pages. You may find that a page will need to manipulate components, or will require additional logic. When that is the case, you can create a backing bean for the page. For more information, see [Section 9.4, "Creating and Using a Backing Bean for a Web Page"](#).

Web pages typically fall into two categories: those used to display data and those used to collect data. In the SOADEMO-CLIENT application, the `ProductsFacadeLocal` data control was used to declaratively create the table that displays products on the `browseItems` JSP, using ADF data binding. For more information, see [Section 9.5, "Displaying Collection and Detail Data Using ADF Data Binding"](#).

The Registration page collects user information and sends it to the external customer service application that is published as the `CustomerService` web service. This page does not use ADF for data binding. Instead, input text fields were created, and the page's corresponding backing bean handles the binding of entered data. For more information, see [Section 9.6, "Collecting Data for Use by a Web Service"](#). This section details both how to use an ADF data control with a web service, and how to invoke a web service manually from a backing bean.

9.2 Creating a Web Application Workspace

JDeveloper provides application templates that enable you to quickly create an application workspace and project structure with the appropriate combination of technologies already specified. When you choose to create an application using a predefined template, JDeveloper generates many of the needed directories and files for you. Using templates is especially helpful in a JSF application, as many of the needed configuration files are generated for you automatically.

9.2.1 How to Create an Application Workspace

The SOADEMO-CLIENT application uses the **Web Application [JSF, EJB]** application template, which creates one project for the data model, and one project for the controller and view (user interface) components in a workspace.

To create a new application workspace in JDeveloper:

1. Right-click the **Applications** node in the Application Navigator and choose **New Application**.
2. In the Create Application dialog, select an application template from the list.

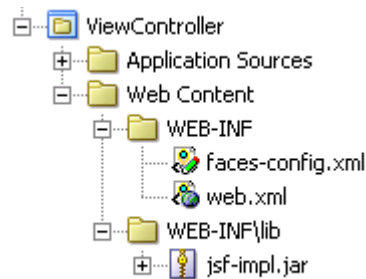
You don't have to use JDeveloper application templates to create an application workspace—they are provided merely for your convenience.

9.2.2 What Happens When You Use an Application Template to Create a Workspace

By default, JDeveloper names the project for the data model `Model`, and the project for the user interface and controller `ViewController`. You can rename the projects using **File > Rename** after you've created them, or you can use **Tools > Manage Templates** to change the default names that JDeveloper uses.

Note: The illustrations and project names used in this chapter are the JDeveloper default names. The SOA Order Booking SOADEMO-CLIENT application, however, uses the project name `UserInterface` for the JSF view and controller components, and `ClientModel` for the project that contains the JPA entities and other classes that work with the Product data. The SOADEMO-CLIENT application also has additional projects in the Application Navigator (for example, `CustomerService` and `OrderService`), which you create manually to organize your application components into logical folders.

[Figure 9–6](#) shows the Application Navigator view of the `ViewController` project after you create the workspace.

Figure 9–6 *ViewController Project in the Navigator After You Create a Workspace*

When you use the **Web Application [JSF, EJB]** template to create a workspace, JDeveloper does the following for you:

- Creates a ViewController project that uses JSF technology. The project properties include:
 - **JSP Tag Libraries:** JSF Core, JSF HTML
 - **Libraries:** JSF, Commons Beanutils, Commons Digester, Commons Logging, Commons Collections, JSTL
 - **Technology Scope:** JSF, JSP and Servlets, Java, HTML

When you work in the ViewController project, the New Gallery will be filtered to show standard web technologies (including JSF) in the Web Tier category.

By default, JDeveloper uses JSTL 1.1 and a J2EE 1.4 web container that supports Servlet 2.4 and JSP 2.0.

- Creates a starter `web.xml` file with default settings in `/WEB-INF` of the ViewController project.
- Creates an empty `faces-config.xml` file in `/WEB-INF` of the ViewController project.
- Adds `jsf-impl.jar` in `/WEB-INF/lib` of the ViewController project.
- Creates a Model project that uses EJB technology. For more information about the Model project, see [Chapter 4, "Building and Using Application Services"](#).

9.3 Creating a Web Page

While JSF supports a number of presentation layer technologies, JDeveloper uses JSP as the presentation technology for creating JSF web pages. When you use JSF with JSP, the JSF pages can be JSP pages (`.jsp`) or JSP documents (`.jspx`). JSP documents are well-formed XML documents, and the XML standard offers many benefits such as validation against a document type definition. Unless otherwise noted, the term *JSF page* in this guide refers to both JSF JSP pages and JSF JSP documents.

9.3.1 How to Create a JSF Web Page

Oracle recommends using the *JSF navigation diagram* to plan out and build your application page flow. Because the JSF navigation diagram visually represents the pages of the application, it is also an especially useful way to drill down into individual web pages when you want to edit them in the JSP/HTML Visual Editor.

To add a JSF page to your ViewController project using the JSF navigation diagram:

1. Expand the `ViewController/Web Content/WEB-INF` folder in the Application Navigator and double-click **faces-config.xml**.

By default, JDeveloper opens the file in the **Diagram** tab, which is the JSF navigation diagram. If you've just started the `ViewController` project, the navigation diagram would be an empty drawing surface. If you don't see a blank drawing surface when you open `faces-config.xml`, select **Diagram** at the bottom of the editor.
2. In the Component Palette, select **JSF Navigation Diagram** from the dropdown list, and then select **JSF Page**.
3. Click on the diagram in the place where you want the page to appear. A page icon with a label for the page name appears on the diagram. The page icon has a yellow warning overlaid. This means you haven't created the actual page yet, just a representation of the page.
4. To create the new page, double-click the page icon and use the Create JSF JSP wizard.

When creating a page in JDeveloper for the first time, be sure to complete all the steps of the wizard.

5. In Step 1 of the Create JSF JSP wizard, select **JSP Page (*.jsp)** or **JSP Document (*.jspx)** for the JSP file **Type**.
6. Enter a filename and accept the default directory name or choose a new location. By default, JDeveloper saves files in `/ViewController/public_html` in the file system.
7. In Step 2 of the wizard, select whether or not to automatically expose UI components in a managed bean.

For more information about managed beans and UI components, see [Section 9.4, "Creating and Using a Backing Bean for a Web Page"](#).
8. In Step 3 of the wizard, make sure that these libraries are added to the **Selected Libraries** list:
 - **ADF Faces Components**
 - **ADF Faces HTML**
 - **JSF Core**
 - **JSF HTML**
9. Accept the default selection for the remaining page and click **Finish**.

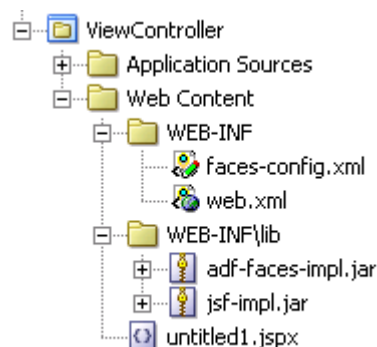
Your new JSF page opens in the JSP/HTML Visual Editor, where you can begin to lay out the page using ADF Faces components from the Component Palette or databound ADF Faces components dropped from the Data Control Palette.

If you switch back to the JSF navigation diagram (by clicking the **faces-config.xml** editor tab at the top), you will notice that the page icon no longer has the yellow warning overlaid.

9.3.2 What Happens When You Create a JSF Web Page

Figure 9–7 shows the Application Navigator view of the ViewController project after you complete the wizard steps to add a JSF page.

Figure 9–7 *ViewController Project in the Navigator After You Add a JSF Page*



JDeveloper does the following when you create your first JSF page in a ViewController project via the JSF navigation diagram:

- Adds `adf-faces-impl.jar` to `/WEB-INF/lib`.
- Adds these libraries to the ViewController project properties:
 - **JSP Tag Libraries:** ADF Faces Components, ADF Faces HTML
 - **Libraries:** JSP Runtime, ADF Faces Runtime, ADF Common Runtime
- Creates the `faces-config.oxd_faces` file in the file system only, for example, in `<JDEV_HOME>/jdev/mywork/Application1/ViewController/model/public_html/WEB-INF`. When you plan out and build your page flow in the JSF navigation diagram, this is the file that holds all the diagram details such as layout and annotations. JDeveloper always maintains this file alongside its associated XML file, `faces-config.xml`. The `faces-config.oxd_faces` file is not visible in the Application or System Navigator.

JDeveloper also creates a starter page that is a JSF JSP 2.0 file, and automatically imports the JSF tag libraries into the starter page. If you select to add the ADF Faces tag libraries in step 3 of the wizard, JDeveloper also imports the ADF Faces tag libraries into the starter pages. Example 9–1 shows a starter page for a JSF JSP document.

Example 9–1 *Starter JSF JSP Document Created by JDeveloper*

```
<?xml version='1.0' encoding='windows-1252'?>
<jsp:root xmlns:jsp="http://java.sun.com/JSP/Page" version="2.0"
          xmlns:f="http://java.sun.com/jsf/core"
          xmlns:h="http://java.sun.com/jsf/html"
          xmlns:af="http://xmlns.oracle.com/adf/faces"
          xmlns:afh="http://xmlns.oracle.com/adf/faces/html"
  <jsp:output omit-xml-declaration="true" doctype-root-element="HTML"
            doctype-system="http://www.w3.org/TR/html4/loose.dtd"
            doctype-public="-//W3C//DTD HTML 4.01 Transitional//EN"/>
  <jsp:directive.page contentType="text/html; charset=windows-1252"/>
  <f:view>
    <html>
```

```
<head>
  <meta http-equiv="Content-Type"
        content="text/html; charset=windows-1252"/>
  <title>untitled1</title>
</head>
<body>
  <h:form></h:form>
</body>
</html>
</f:view>
</jsp:root>
```

9.3.3 What You May Need to Know About Using the JSF Navigation Diagram

In the JSF navigation diagram, you will notice that the label of the page icon has an initial slash (/), followed by the name of the page. The initial slash is required so that the page can be run from the diagram. If you remove the slash, JDeveloper will automatically reinstate it for you.

Be careful when renaming and deleting pages from the JSF navigation diagram:

- **Renaming pages:** If you rename a JSF page on a JSF navigation diagram, this is equivalent to removing a page with the original name from the diagram and adding a new one with the new name: the page icon changes to a page icon overlaid with the yellow warning, indicating that the page does not yet exist. If you have already created the underlying page, that page remains with its original name in the Application Navigator.

Similarly, if you have a JSF page in the Application Navigator and the page icon is displayed on the diagram, if you now rename the page in the Application Navigator, this is equivalent to removing the original file and creating a new file. The diagram, however, retains the original name, and now displays the page icon overlaid with the yellow warning, indicating that the page does not exist.

- **Deleting pages:** When you delete a page icon in the JSF navigation diagram, the associated web page is no longer visible in the diagram. If you have created the actual file, it is still available from the `Web Content` folder in the ViewController project in the Application Navigator.

For information about the JSF navigation diagram and creating navigation, see *Oracle Application Development Framework Developer's Guide*.

9.4 Creating and Using a Backing Bean for a Web Page

In JSF, backing beans are JavaBeans used mainly to provide UI logic and to manage data between the web tier and the business tier of the application (similar to a data transfer object). Typically you have one backing bean per JSF page. The backing bean contains the logic and properties for the UI components used on the page. For example, to programmatically change a UI component as a result of some user activity or to invoke a service method, you provide the necessary code in the page's backing bean and bind the component to the corresponding property or method in the bean.

For example, in the SOADEMO-CLIENT application, the `Register` backing bean contains a method to which the `Register` command component is bound. When the user clicks the **Register** button, the data entered on the page is submitted, and this method is invoked. Because the data from the submitted fields is bound to instances of the components using the backing bean, the method can access those values to create a `Customer` object.

For a backing bean to be available when the application starts, you register it as a managed bean with a name and scope in `faces-config.xml`. At runtime, whenever the bean is referenced on a page through a JSF EL value or method binding expression, the JSF implementation automatically instantiates the bean, populates it with any declared, default values, and places it in the managed bean scope as defined in `faces-config.xml`.

9.4.1 How to Create and Configure a Backing Bean

The Overview mode of the JSF Configuration Editor lets you create and configure a backing bean declaratively.

To create and configure a backing bean as a managed bean:

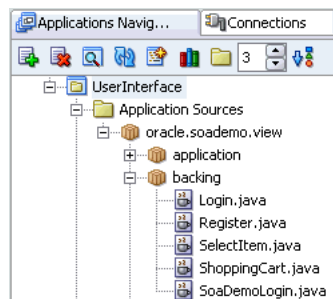
1. In the Application Navigator, double-click `faces-config.xml` to open it in the default mode of the JSF Configuration Editor.
2. At the bottom of the editor, select the **Overview** tab to switch to the Overview mode, if necessary.
3. In the element list on the left, select **Managed Beans**.
4. Click **New** to open the Create Managed Bean dialog.
5. In the dialog, specify the following for a managed bean:
 - **Name:** Enter a unique identifier for the managed bean (e.g., `SOA_Demo_Login`). This identifier determines how the bean will be referred to within the application using EL expressions, instead of using the bean's fully qualified class name.
 - **Class:** Enter the fully qualified class name (e.g., `oracle.srdemo.view.backing.SoaDemoLogin`). This is the JavaBean that contains the properties that hold the data for the UI components used on the page, along with the corresponding accessor methods and any other methods (such as navigation or validation). This can be an existing or a new class.
 - **Scope:** This determines the scope within which the bean is stored. The valid scope values are:
 - **application:** The bean is available for the duration of the web application. This is helpful for global beans such as LDAP directories.
 - **session:** The bean is available to the client throughout the client's session.
 - **request:** The bean is available from the time it is instantiated until a response is sent back to the client. This is usually the life of the current page. Backing beans for pages usually use this scope.
 - **none:** The bean is instantiated each time it is referenced.
6. Select the **Generate Class If It Does Not Exist** checkbox to let JDeveloper create the Java class for you. If you've already created the Java class, don't select this checkbox.

Note: At this point, you haven't defined a strict relationship between the JSF page and the backing bean. You've simply configured a backing bean in `faces-config.xml`, which you can now reference via JSF EL expressions on a page.

9.4.2 What Happens When You Create and Configure a Backing Bean

If you select the **Generate Class If It Does Not Exist** checkbox, JDeveloper creates a new Java class using the fully qualified class name set as the value of **Class**. The new file appears within the **Application Sources** node of the `ViewController` project in the Application Navigator, as illustrated in [Figure 9–8](#).

Figure 9–8 Backing Bean for SOALogin page in the Navigator



To edit the backing bean class, double-click the file in the Application Navigator (for example, `SoaDemoLogin.java`) to open it in the source editor. If it's a new class, you would see something similar to [Example 9–2](#).

Example 9–2 Empty Java Class Created by JDeveloper

```
package oracle.soademo.view.backing;

public class SoaDemoLogin {
    public SoaDemoLogin() {
    }
}
```

In `faces-config.xml`, JDeveloper adds the backing bean configuration using the `<managed-bean>` element, as shown in [Example 9–3](#).

Example 9–3 Registering a Managed Bean in the `faces-config.xml` File

```
<faces-config xmlns="http://java.sun.com/JSF/Configuration">
    ...
    <!-- Page backing beans typically use request scope-->
    <managed-bean>
        <managed-bean-name>SOA_Demo_Login</managed-bean-name>
        <managed-bean-class>oracle.srdemo.view.backing.SoaDemoLogin</managed-bean-class>
        <managed-bean-scope>request</managed-bean-scope>
    </managed-bean>
    ...
</faces-config>
```

9.4.3 How to Use the Automatic Component Binding Feature

JDeveloper has a feature that lets you automatically bind a UI component instance on a JSF page to a backing bean property. When you turn on the Auto Bind feature for a page, for any UI component that you insert into the page, JDeveloper automatically adds property code in the page's backing bean, and binds the component's binding attribute to the corresponding property in the backing bean. If your backing bean doesn't have to modify the attributes of UI components on a page programmatically, you don't need to use the automatic component binding feature.

You can decide to use automatic component binding when creating a JSF page, as shown in Step 7 in [Section 9.3.1, "How to Create a JSF Web Page"](#). You can also manually turn on automatic component binding for an existing page.

To turn on automatic component binding for an existing JSF page:

1. Open the JSF page in the visual editor. Select **Design** at the bottom of the editor window.
2. Choose **Design > Page Properties** to display the Page Properties dialog.
3. Select **Component Binding**.
4. Select **Auto Bind**.
5. Select a managed bean from the dropdown list or click **New** to configure a new managed bean for the page.

Note: By turning on automatic component binding in a JSF page, you are defining a strict relationship between a page and a backing bean in JDeveloper.

9.4.4 What Happens When You Use Automatic Component Binding in JDeveloper

If the Auto Bind feature is turned on for a JSF page, you'll see a special comment line near the end of the page:

```
...
    </f:view>
    <!--oracle-jdev-comment:auto-binding-backing-bean-name:backing_soaDemoPage-->
</jsp:root>
```

In `faces-config.xml`, a similar comment line is inserted at the end of the page's backing bean configuration:

```
<managed-bean>
  <managed-bean-name>backing_soaDemoPage</managed-bean-name>
  <managed-bean-class>oracle.srdemo.view.backing.soaDemoPage</managed-bean-class>
  <managed-bean-scope>request</managed-bean-scope>
  <!--oracle-jdev-comment:managed-bean-jsp-link:1soaDemoPage.jsp-->
</managed-bean>
```

When you turn on the Auto Bind feature for a page, JDeveloper does the following for you every time you add a UI component to the page:

- Adds a property and property accessor methods for the component in the backing bean. For example, the next code snippet shows the code added for an `inputText` and a `commandButton` component:

```
private CoreInputText inputText1;
private CoreCommandButton commandButton1;
public void setInputText1(CoreInputText inputText1) {
    this.inputText1 = inputText1;
}

public CoreInputText getInputText1() {
    return inputText1;
}
```

```

public void setCommandButton1(CoreCommandButton commandButton1) {
    this.commandButton1 = commandButton1;
}

public CoreCommandButton getCommandButton1() {
    return commandButton1;
}

```

- Binds the component to the corresponding bean property using an EL expression as the value for the binding attribute, as shown in this code snippet:

```

<af:inputText binding="#{backing_soaDemoPage.inputText1}"
<af:commandButton binding="#{backing_soaDemoPage.commandButton1}"

```

When you turn off the Auto Bind feature for a page, JDeveloper removes the special comments from the JSF page and `faces-config.xml`. The binding EL expressions on the page and the associated backing bean code are not deleted.

Tip: When Auto Bind is turned on and you delete a UI component from a page, JDeveloper automatically removes the corresponding property and accessor methods from the page's backing bean.

9.5 Displaying Collection and Detail Data Using ADF Data Binding

When you need to display data on a Web page, ADF data binding allows you to easily and declaratively create a table bound to data using ADF Faces components. By default, users will be able to scroll through row sets, and select a row. You then can create a new page that shows the details of the selected row, and add the navigation between the two pages. ADF data binding will automatically display the details of the selected row, without need for any additional code.

For example, [Figure 9–9](#) shows the `BrowseItems` page in the `SOADEMO-CLIENT` application. Users can select a product then click the **View Details** button to navigate to the `SelectItem` page, where they can view details for the selected item.

Figure 9–9 The BrowseItems Page Allows Users to Select a Product

Browse and Select Items

Select and View Details				
			Previous	1-10 of 27
Select	name	prodid	description	category
☒	HD Television	128723	A High-Definition Flat Screen Television that uses DLP Technology.	Video
☐	PlayStation 2 Video Game System	223453	Playstation II Interactive Video Gaming System.	Games
☐	Treo 650 Phone/PDA	429832	A fully featured Treo Phone and PDA combination.	Handhelds
☐	Treo 700w Phone/PDA	423322	A fully featured Treo Phone and Windows CE Handheld.	Handhelds
☐	Tungsten E PDA	592039	Tungsten handheld Personal Digital Assistant.	Handhelds
☐	XBox Video Game System	621135	XBox Interactive Video Gaming System	Games
☐	XBox 360 Video Game System	218407	Next generation Xbox 360 Interactive Video Gaming System.	Games
☐	Playstation Portable	124039	Playstation Portable HandHeld Video Gaming System.	Handhelds
☐	Nintendo DS	606945	Nintendo DS Handheld Gaming System	Handhelds
☐	Razer Cellular Phone	927384	Ultra-sleek Razer Cellular Phone	Handhelds

[Figure 9–10](#) shows the `SelectItem` page for the selected product.

Figure 9–10 The SelectItem Page Allows Users to View Details and Add Item to Cart.

The screenshot shows a web page titled "Item Details". On the left, there is a table of item information:

name	HD Television
description	A High-Definition Flat Screen Television that uses DLP Technology.
category	Video
listPrice	\$1999.00
prodid	128723
producer	Azimuth

Below the table is a "Continue Shopping" button. To the right of the table is a form titled "Add to cart?". It includes a "Quantity?" label with a dropdown menu, an "Add to cart" button, and a "Go to Shopping Cart" link at the bottom.

To create a page that displays a collection of data, and then link to a page that displays the details of the selected item, you need to:

1. Create a navigation diagram for the two pages.
2. On the page that will display the collection of items (the browse page), create an ADF Faces table to display the collection and add a button to navigate to the details page.
3. On the details page, create a form that displays the details.

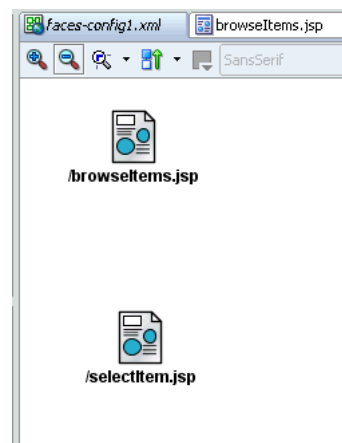
9.5.1 How to Create Navigation Between Pages

Creating a navigation diagram allows you to both create pages and create the navigation between them.

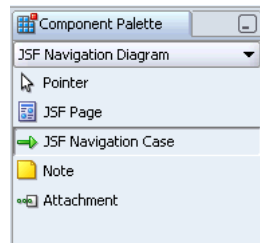
To create a navigation diagram:

1. Follow the steps for creating a web page, as documented in [Section 9.3.1, "How to Create a JSF Web Page"](#) to create both the browse page and the detail page.

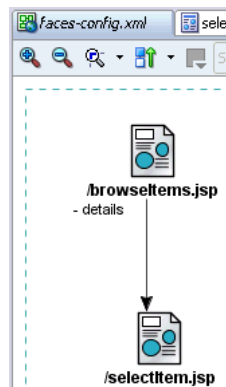
For example, for the SOADEMO-CLIENT, you would create a JSP for the `BrowseItems` page and the `SelectItem` page, as shown in [Figure 9–11](#).

Figure 9–11 JSP Pages on the Navigation Diagram

2. From the Component Palette click the **JSF Navigation Case** component (as shown in [Figure 9–12](#)), click on the `browseItems.jsp` icon and then the `selectItem.jsp` icon, to create a new navigation case between the two pages.

Figure 9–12 Navigation Case in the Component Palette

By default, the outcome value is set to success, as shown in [Figure 9–13](#).

Figure 9–13 Navigation Outcome

3. Double-click the `success` outcome text, and change it something more meaningful, such as `details`.
4. Create another navigation case that goes from the `selectItem` page back to the `browseItems` page. Rename this outcome value to something more meaningful, such as `back`.

Tip: You can easily create more readable diagrams by dragging the JSF Navigation Case line to create angles.

Now that the navigation cases are created, you can place a command component on the first page, set the action attribute to the value of the outcome, and the user will navigate to the second page. For example, if you place a command button on the `browseItems` page and set its action attribute to `details`, the user will navigate to the `selectItem` page.

9.5.2 What Happens When You Create Navigation

When you create a navigation rule using the JSF Navigation Modeler or the JSF Configuration Editor, JDeveloper automatically adds the navigation rule elements to the `faces-config.xml` file for you.

[Example 9–4](#) shows two navigation rules defined in the `faces-config.xml` file. The first rule navigates from the `browseItems` page to the `selectItem` page when the outcome specified in the action attribute of an activated navigation component is `details`. The second rule navigates from the `selectItem` page back to the `browseItems` page when the action attribute of an activated navigation component is `return`.

Example 9-4 Navigation Rules in the faces-config.xml File

```

<navigation-rule>
  <from-view-id>/retailclient/browseItems.jsp</from-view-id>
  <navigation-case>
    <from-outcome>details</from-outcome>
    <to-view-id>/retailclient/selectItem.jsp</to-view-id>
  </navigation-case>
</navigation-rule>
<navigation-rule>
  <from-view-id>/retailclient/selectItem.jsp</from-view-id>
  <navigation-case>
    <from-outcome>return</from-outcome>
    <to-view-id>/retailclient/browseItems.jsp</to-view-id>
  </navigation-case>
</navigation-rule>

```

9.5.3 What Happens at Runtime

The Sun JSF Reference Implementation reads the navigation rules in the `faces-config.xml` file and calls the `NavigationHandler` class, which evaluates the navigation rules and determines which page to display. The handler first finds all rules for the page from which the navigation is originating. It then checks the outcome either returned by an action method or set as the outcome string for the action property of the component, to determine the correct rule to use.

9.5.4 What You May Need to Know About the Navigation Modeler

When using the navigation modeler to create and maintain page navigation, be aware of the following features:

- Changes to navigation rules made directly in the `faces-config.xml` file using the XML editor or made in the configuration editor usually refresh the navigation modeler. Each JSF configuration file has its own navigation modeler diagram. If the information in a navigation diagram does not match the information in its `faces-config.xml` file, you can manually refresh the diagram by right-clicking within the diagram and choosing **Diagram > Refresh diagram from faces-config**.
- When you delete a navigation case on the diagram, the associated `navigation-case` element is removed from the `faces-config.xml` file. If you remove all the cases in a rule, the `navigation-rule` element remains in the `faces-config.xml` file. You can remove the rule directly in the `faces-config.xml` file.
- When you edit the label for the navigation case on the diagram, the associated `navigation-case` element is updated in the `faces-config.xml` file. You cannot change the destination of the navigation case in the diagram. You can, however, change the destination of a navigation case in the JSF Configuration Editor or directly in the `faces-config.xml` file itself.
- When you delete a page icon from the navigation diagram, the associated page file is not deleted from the **Web Content** folder in the `ViewController` project in the Application Navigator.
- When you edit pages manually, JDeveloper does not automatically update the navigation diagram or the associated `faces-config.xml` file. Conversely, when you make changes to a page flow that affect the behavior of an existing page, JDeveloper does not automatically update the code in the page. To coordinate the

navigation diagram with web page changes, right-click within the page in the navigation diagram and choose **Diagram > Refresh Diagram from All Pages**.

- The navigation modeler diagram is the default editor for the `faces-config.xml` file. If you have a large or complex application, loading the diagram may be slow, because the file may be large. If you do not want JSF diagram files to be created for your JSF configuration files, use the **Tools > Preferences > File Types > Default Editor > JSF Configuration File** option to change the default editor. If you change the default editor before opening the `faces-config.xml` file for the first time, no diagram file is created unless you specifically request one.

9.5.5 How to Create a Table to Display Collection Data on a Page

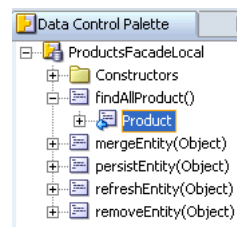
You can display an entire collection of data by dragging the returned collection of a method represented on a data control onto a JSF page as a table.

To add collection data to a page:

1. Open a JSF page in the visual editor.
2. From the Data Control Palette, select a method return that returns a collection.

For example, to create a table that displays all products, you drag the returned `Product` collection of the `findAllProduct` method. [Figure 9–14](#) shows the `Product` collection in the Data Control Palette.

Figure 9–14 Method Return



3. Drag the method return onto the JSF page, and from the context menu, choose the appropriate table.

When you drag the collection, you can choose from the following types of tables:

- **ADF Table:** Allows you to select the specific attributes you wish your editable table columns to display, and what UI components to use to display the data. By default, each attribute on the collection object is displayed in an `inputText` component, thus enabling the table to be editable.
- **ADF Read-Only Table:** Same as the **ADF Table**; however, each attribute is displayed in an `outputText` component.
- **ADF Read-Only Dynamic Table:** The attributes returned and displayed are determined dynamically. This component is helpful when the attributes for the corresponding object are not known until runtime, or you do not wish to hardcode the column names in the JSF page. For example, if you have a method that returns a polymorphic collection (i.e. `getAnimals()` can return a collection of mammals or a collection of birds), the dynamic table can display the different attributes accordingly.

Note: You can also drop a collection as an ADF Master Table, Inline Detail Table. For more information, see *Oracle Application Development Framework Developer's Guide*.

4. From the ensuing Edit Table Columns dialog, you can do the following:

- Change the display label for a column. By default, the label is bound to the `label` property of the table binding. This binding to the property allows you to change the value of the label text once and have it appear the same on all pages that display the label. In this dialog, you can instead enter text or an EL expression to bind the label value to something else, for example, a key in a resource file.

For example, the heading for the description column in the table on the `browseItems` page is bound to the label property of the corresponding attribute binding:

```
#(bindings.findAllProduct1.labels.description)
```

- Change the attribute binding for a column.

For example, you can change the description column to instead be bound to the `listPrice` attribute. Note the following:

- If you change the binding, the label for the column also changes.
- If you change the binding to an attribute currently bound to another column, the UI component changes to a component different from that used for the column currently bound to that attribute.

If you simply want to rearrange the columns, you should use the order buttons. See the fourth bullet point below for more information.

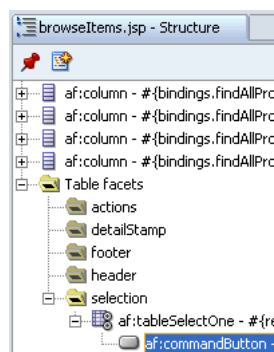
- Change the UI component used to display an attribute. The UI components are either `inputText` or `outputText` and are set based on the table you selected when you dropped the collection onto the page. You can change to the other component using the dropdown menu. If you want to use a different component, such as a command link or button, you need to use this dialog to select the `outputText` component, and then in the Structure window, add that other UI component (such as a command link) as a parent to this component.
- Change the order of the columns using the order buttons.
- Add a column using the **New** button. Doing so adds a new column at the bottom of the dialog and populates it by default with values from the next sequential attribute in the collection. You then need to edit the values. You can only select an attribute associated with the object to which the table is bound.
- Delete a column using the **Delete** button. Doing so deletes the column from the table.
- Add a `tableSelectOne` component to the table's `selection` facet by selecting **Enable selection**. In order to navigate to a page that displays details for a selected row in the table, you must select this option.
- Allow sorting for all columns by selecting **Enable sorting**.

Note: If you choose to enable sorting, the table can only sort through the number of objects returned by the iterator, as determined by the iterator's `rangeSize` attribute. For more information about iterators and the `rangeSize` attribute, refer to the *Oracle Application Development Framework Developer's Guide*.

5. To add a command button that will navigate to the details page, in the Structure window, select the `commandButton` for the `tableSelectOne` component, as shown in [Figure 9–15](#).

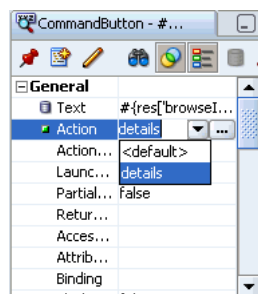
Note: This component will only be present if you selected the **Enable Selection** option when you created the table in the previous step.

Figure 9–15 *Command Button in a `tableSelectOne` Component*



In the Property Inspector, use the `Action` attribute's dropdown menu to choose the outcome created in the navigation diagram in the previous set of steps. For example, if you created an outcome named `details` between the two pages, you would select **details**. [Figure 9–16](#) shows the Property Inspector with this outcome selected for the `Action` attribute.

Figure 9–16 *Action Options in the Property Inspector*



Now when users click this button, they will navigate to the page defined in the navigation case, in this case, the `selectItem` page.

9.5.6 What Happens When You Add a Collection to a Page

When you create an ADF web application by dragging and dropping items from the Data Control Palette, JDeveloper creates and registers all the files needed to run the application. Specifically, JDeveloper:

- Creates a `DataBindings.cpx` file in the view package in the **Application Sources** directory (if one does not already exist), and adds an entry for any pages.

The `DataBindings.cpx` file defines the binding context for the application. It maps individual pages to their corresponding page definition files and registers the data controls used by those pages. The data controls are defined in the `DataControls.dcx` file.

- Registers the ADF binding filter in the `web.xml` file.

The ADF binding filter preprocesses any HTTP requests that may require access to the binding context.

- Registers the ADF phase listener in the `faces-config.xml` file.

- Adds the following ADF runtime libraries to the project properties of the view project:

- ADF Model Runtime (`adfm.jar`)
- ADF Controller (`adf-controller.jar`)

- Adds a page definition file (if one does not already exist for the page) to the page definition subpackage, the name of which is defined in the ADFm settings of the project properties. The default subpackage is `view.pageDefs` in the `Application Sources` directory.

The page definition file (`<pageName>PageDef.xml`) defines the ADF binding container for each page in an application's view layer. The binding container provides runtime access to all the ADF binding objects.

- Configures the page definition file, which includes adding definitions of the binding objects referenced by the page.

[Example 9–5](#) shows the page definition file created when you drop the `Product` collection. The method iterator references the return of the `findAllProduct` method. This is what allows the page to access data for every instance returned. The method action binding is what actually invokes the method. The table binding references the results in the iterator, and allows the data for each attribute to be displayed.

Example 9–5 Page Definition Code

```
<executables>
  <methodIterator id="findAllProductIter" Binds="findAllProduct.result"
    DataControl="ProductsFacadeLocal" RangeSize="10"
    BeanClass="oracle.soademo.clientmodel.Product"/>
</executables>
<bindings>
  <methodAction id="findAllProduct"
    InstanceName="ProductsFacadeLocal.dataProvider"
    DataControl="ProductsFacadeLocal" MethodName="findAllProduct"
    RequiresUpdateModel="true" Action="999"
    ReturnName="ProductsFacadeLocal.methodResults.ProductsFacadeLocal_
      dataProvider_findAllProduct_result"/>
```

```

<table id="findAllProduct1" IterBinding="findAllProductIter">
  <AttrNames>
    <Item Value="category"/>
    <Item Value="description"/>
    <Item Value="id"/>
    <Item Value="listPrice"/>
    <Item Value="name"/>
    <Item Value="prodid"/>
    <Item Value="producer"/>
  </AttrNames>
</table>
</bindings>

```

- Adds prebuilt components to the JSF page.

When you drop a table onto a JSF page, JDeveloper creates a table that contains a column for each attribute on the object to which it is bound. To do this, JDeveloper inserts an ADF Faces table component. This component contains an ADF Faces column component for each attribute named in the table binding. Each column then contains either an input or outputText component bound to the attribute's value. Each column's heading attribute is bound to the label property for each attribute on the table binding. [Example 9–6](#) shows some of the code generated when you drop the `Product` return as a table.

Example 9–6 JSF Page Code

```

<af:table value="#{bindings.findAllProduct1.collectionModel}" var="row"
  rows="#{bindings.findAllProduct1.rangeSize}"
  first="#{bindings.findAllProduct1.rangeStart}"
  emptyText="#{bindings.findAllProduct1.viewable ? \
    'No rows yet.' : 'Access Denied.'}">
  <af:column headerText="#{bindings.findAllProduct1.labels.category}"
    sortProperty="category" sortable="false">
    <af:outputText value="#{row.category}"/>
  </af:column>
  <af:column headerText="#{bindings.findAllProduct1.labels.description}"
    sortProperty="description" sortable="false">
    <af:outputText value="#{row.description}"/>
  </af:column>
  ...
</af:table>

```

An ADF Faces table itself iterates over the data accessed by the iterator binding. In order to do this, the table wraps the result set from the iterator binding in an `oracle.adf.view.faces.model.CollectionModel` object. As the table iterates, it makes each item in the collection available within the table component using the `var` attribute.

In [Example 9–6](#), the table iterates over the collection from the `findAllProduct1` table binding, which in turn references the `findAllProductIter` iterator binding. The iterator binding is what determines the current data object. When you set the `var` attribute on the table to `row`, each column then accesses the current data object for the current row presented to the table tag using the `row` variable, as shown for the value of the `outputText` tag:

```

<af:outputText value="#{row.category}"/>

```

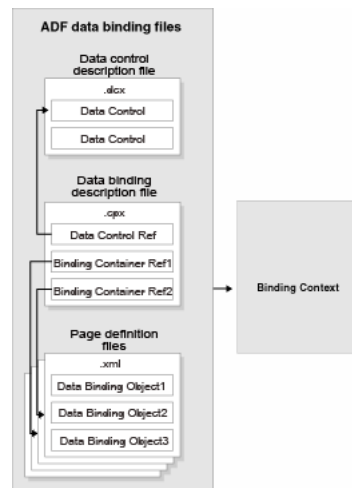
- Adds all the libraries, files, and configuration elements required by ADF Faces components, if ADF Faces components are used.

9.5.7 What Happens at Runtime

When a page contains ADF bindings, at runtime the interaction with the business services initiated from the client or controller is managed by the application through a single object known as the Oracle ADF binding context. The ADF binding context is a container object that contains a list of data controls and data binding objects derived from the Oracle ADF Model layer.

The ADF life cycle creates the Oracle ADF binding context from the `DataControls.dcx`, `DataBindings.cpx`, and page definition files, as shown in [Figure 9-17](#). The `DataControls.dcx` file defines all the data controls available to the application. The `DataBindings.cpx` file references the data controls that are currently being used by pages in the application and maps the binding containers, which contain the binding objects defined in the page definition files, to web page URLs. The page definition files define the binding objects used by the application pages. There is one page definition for each page.

Figure 9-17 ADF Binding File Runtime Usage



Using the table created from the `Product` return as an example, the binding container invokes the `findAllProductIter` method iterator, which in turn invokes the `findAllProduct` method that returns the `Product` collection. The iterator then iterates over the data and makes the data for the first found record available to the UI component by placing it in the binding container. Because there is a binding for the `findAllProduct1` table in the page definition that can access the value from the iterator (see [Example 9-5](#)), and since the UI component is bound to the `findAllProduct1` binding using an EL expression `{bindings.findAllProduct1.collectionModel}`, that data is displayed by that component.

For more information about what specifically happens at runtime with ADF tables, refer to the *Oracle Application Development Framework Developer's Guide*.

9.5.8 How to Display Details on a Page

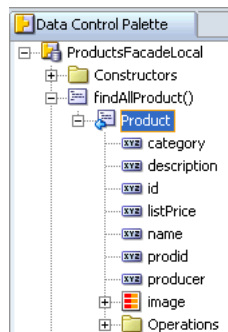
You can display the detailed data of an item returned by a method by dragging the returned collection as a form. By using a read-only form, only the currently selected data will be displayed.

To add a form to display details of an item:

1. Open a JSF page in the visual editor.
2. From the Data Control Palette, select a method return that returns a collection.

For example, to create a form that displays information about a product, you drag the product return from the `findAllProduct` method, as shown in [Figure 9–18](#). This method creates a form with data already populated in the text fields.

Figure 9–18 Attributes Associated with a Returned Collection in the Data Control Palette



3. Drag the collection onto the page, and from the context menu choose the type of form to display data for the object. For a form, you are given the following choices:
 - **ADF Form:** Launches the Edit Form Fields dialog that allows you to select individual attributes instead of creating a field for every attribute by default. It also allows you to select the label and UI component used for each attribute. By default, ADF `inputText` components are used, except for dates, which use the `selectInputDate` component. Each `inputText` component contains a validator tag that allows you to set up validation for the attribute. You can elect to include navigational controls that allow users to navigate through all the data objects in the collection. For additional help in using the dialog, click **Help**. All UI components are placed inside a `panelForm` component.
 - **ADF Read-Only Form:** Same as the ADF Form, but by default, `outputText` components are used. Since the form is meant to display data, no validator tags are added. The `label` attribute is populated for each component. Attributes of type `Date` also use the `outputText` component. All components are placed inside `panelLabelAndMessage` components, which are in turn placed inside a `panelForm` component.

Note: You can also drop a collection as an ADF Creation Form. For more information, see *Oracle Application Development Framework Developer's Guide*.

The form will now display the data for the item selected in the table.

9.5.9 What Happens When You Display Detail Data

Dropping a form from the Data Control Palette has the same effect as dropping a table. For more information, see [Section 9.5.6, "What Happens When You Add a Collection to a Page"](#). Briefly, JDeveloper does the following:

- Creates the bindings for the form and adds the bindings to the page definition file. [Example 9-7](#) shows the page definition code created when you drop the `Product` return as a form.

Example 9-7 Page Definition Code for a Form

```
<executables>
  <methodIterator id="findAllProductIter" Binds="findAllProduct.result"
                 DataControl="ProductsFacadeLocal" RangeSize="10"
                 BeanClass="oracle.soademo.clientmodel.Product"/>
</executables>
<bindings>
  <methodAction id="findAllProduct"
                InstanceName="ProductsFacadeLocal.dataProvider"
                DataControl="ProductsFacadeLocal" MethodName="findAllProduct"
                RequiresUpdateModel="true" Action="999"
                IsViewObjectMethod="false"
                ReturnName="ProductsFacadeLocal.methodResults.ProductsFacadeLocal_
                           dataProvider_findAllProduct_result"/>
  <attributeValues id="category" IterBinding="findAllProductIter">
    <AttrNames>
      <Item Value="category"/>
    </AttrNames>
  </attributeValues>
  <attributeValues id="description" IterBinding="findAllProductIter">
    <AttrNames>
      <Item Value="description"/>
    </AttrNames>
  </attributeValues>
  ...
</bindings>
```

Note that the method iterator and the method binding are the same for the table as for the form. However, instead of using a table binding, each attribute has its own binding.

- Adds the necessary code for the UI components to the JSF page.

The attributes on the UI components (such as value) are bound to properties on that attribute's binding object (such as `inputValue`). [Example 9-8](#) shows the code generated on the JSF page when you drop the `Product` collection for the `findAllProduct` method as a default ADF Form.

Example 9-8 JSF Code for an ADF Form

```
<af:panelForm>
  <af:panelLabelAndMessage label="#{bindings.category.label}">
    f:outputText value="#{bindings.category.inputValue}"/>
  </af:panelLabelAndMessage>
  <af:panelLabelAndMessage label="#{bindings.description.label}">
    <af:outputText value="#{bindings.description.inputValue}"/>
  </af:panelLabelAndMessage>
  <af:panelLabelAndMessage label="#{bindings.id.label}">
    <af:outputText value="#{bindings.id.inputValue}">
      <f:convertNumber groupingUsed="false"
                       pattern="#{bindings.id.format}"/>
    </af:outputText>
  </af:panelLabelAndMessage>
  ...
</af:panelForm>
```

9.6 Collecting Data for Use by a Web Service

In an SOA system, you may need to collect data that will be used by an external application. For example, in the Order Booking system, the SOADEMO-CLIENT application collects customer information, which is then passed to the external customer service application, which, in turn, creates a new customer.

Applications and components within an SOA system must expose public service interfaces, typically using WSDL. The web client may therefore either interact with the application components (via RMI) or with the web service interfaces. When interacting through a web service, you can determine the data that a page may need to collect (for example, the attribute names and types) by using the provided WSDL to create a service proxy to the web service.

If methods on the service take simple data types as parameters, then you can create forms that collect data using ADF data binding, similar to the forms created to display data. First, you create a data control based on the provided WSDL. For the procedure, see [Section 5.3.3, "How to Create a Data Control From a Web Service"](#). Then, you drop the method itself (as opposed to the return of a method), to create the input form, as described in the following section.

If a method takes a complex object as a parameter, instead of using ADF data binding, you must create the logic to collect the data and pass it to the service. You add this logic to the page's backing bean. For the procedure, see [Section 9.6.3, "How to Invoke a Service Using a Backing Bean"](#).

9.6.1 How to Create an Input Form Using a Web Service Data Control

Instead of dragging a return from a method, as you do to display existing data, you drag the method itself to create a parameterized form.

Note: You cannot successfully use a data control to create complex objects. For example, to create a `Customer` object for the Order Booking application, you must also create an `Address` object. To do this, you must manually invoke the service from a backing bean. For more information, see [Section 9.6.3, "How to Invoke a Service Using a Backing Bean"](#).

To create an ADF input form:

1. Open a JSF page in the visual editor.
2. From the Data Control Palette, drag the appropriate operation onto the JSF page.
3. From the context menu, choose **Parameters > ADF Parameter Form**.

The Edit Form Fields dialog opens, which allows you to customize the labels, bindings, and UI components before creating the form. JDeveloper automatically adds a command button bound to the method, which will invoke the web service and send the data captured in the form.

9.6.2 What Happens When You Create an ADF Input Form

When you drop an operation as a parameter form, JDeveloper:

- Defines variables to hold the data values, a method binding for the operation, and the attribute bindings for the associated attributes in the page definition file.
- Inserts code in the JSF page for the form using ADF Faces `inputText` components and an ADF Faces command button component. This code is the same as code for any other input form or command button.

When you create a parameter form, JDeveloper creates `NamedData` elements for each parameter. Because the user will provide the parameter values, each `NamedData` element is bound to the attribute binding for the corresponding attribute. This binding allows the operation to access the attribute's value for the parameter on execution.

Note that like the attributes for a collection, attributes for a method also reference an iterator. However, instead of referencing a method iterator that accesses and iterates over the collection that the associated method returns, attributes for a creation-type method access and iterate over variables. Because this type of method has not returned an object, there is nothing to hold the values entered on the page. Variables act as the data holders.

JDeveloper creates a variable for each parameter the method takes. The variables are declared as children to the variable iterator, and are local, meaning they "live" only as long as the associated binding context. [Example 9-9](#) shows the variable iterator and variables created when you use the `addNewCustomer` operation from a data control created from the `CustomerSvc` WSDL. Note that for the purposes of demonstration, not all attributes were used to create the form.

Example 9-9 Variables and the Variable Iterator

```
<executables>
  <variableIterator id="variables">
    <variable Type="java.lang.String" Name="addNewCustomer_customer_fname"
      IsQueryable="false"/>
    <variable Type="java.lang.String" Name="addNewCustomer_customer_lname"
      IsQueryable="false"/>
    <variable Type="java.lang.String" Name="addNewCustomer_customer_email"
      IsQueryable="false"/>
  </variableIterator>
</executables>
```

When the user enters data and submits the form, the variables are populated and the attribute binding can then provide the value for the method's parameters. Note in [Example 9-10](#) that the method parameters are the `NamedData` elements, which use an EL expression that evaluates to the attribute binding. The attribute bindings refer to the variables for their value.

Example 9–10 Method Parameters Use NamedData Elements to Provide Values

```

<bindings>
  <methodAction id="addNewCustomer" MethodName="addNewCustomer"
    RequiresUpdateModel="true" Action="999"
    IsViewObjectMethod="false" DataControl="CustomerSvc"
    InstanceName="CustomerSvc"
    ReturnName="CustomerSvc.methodResults.CustomerSvc_addNewCustomer_result">
    <NamedData NDName="customer_lname" NDType="java.lang.String"
      NDValue="{bindings.addNewCustomer_customer_lname}" />
    <NamedData NDName="customer_fname" NDType="java.lang.String"
      NDValue="{bindings.addNewCustomer_customer_fname}" />
    <NamedData NDName="customer_email" NDType="java.lang.String"
      NDValue="{bindings.addNewCustomer_customer_email}" />
  </methodAction>
  <attributeValues id="customer_fname" IterBinding="variables">
    <AttrNames>
      <Item Value="addNewCustomer_customer_fname" />
    </AttrNames>
  </attributeValues>
  <attributeValues id="customer_lname" IterBinding="variables">
    <AttrNames>
      <Item Value="addNewCustomer_customer_lname" />
    </AttrNames>
  </attributeValues>
  <attributeValues id="customer_email" IterBinding="variables">
    <AttrNames>
      <Item Value="addNewCustomer_customer_email" />
    </AttrNames>
  </attributeValues>
</bindings>

```

For more information about using ADF Parameter forms, refer to the *Oracle Application Development Framework Developer's Guide*.

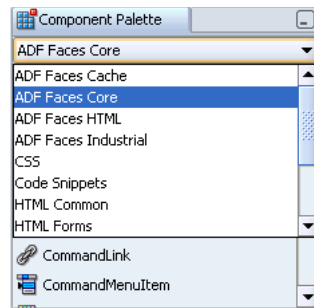
9.6.3 How to Invoke a Service Using a Backing Bean

There may be cases when you will need to directly invoke a web service, and you cannot use a data control. You create the logic to do this in the page's backing bean. For example, in the SOADEMO-CLIENT application, all the logic to collect the data and then send it to the service is on the `Register.java` backing bean.

In order to easily bind the components on a page, you can use the Automatic Component Binding feature when creating the JSF page. This will automatically bind each component to a property in the backing bean. You'll then add logic to the method bound to the command component, which will send the data to the web service.

To invoke a service without using ADF Data Binding:

1. Create a JSF page, as described in [Section 9.3.1, "How to Create a JSF Web Page"](#). However, for step 7 of that procedure, be sure to select the **Automatically Expose UI Components in a New Managed Bean** option. Click **Help** for more information on this step of the wizard.
2. Use the Component Palette's dropdown menu to choose ADF Faces Core, as shown in [Figure 9–19](#). This will allow you to drag ADF Faces components onto the JSF page.

Figure 9–19 Selecting a Tag Library in the Component Palette

3. From the Component Palette, drag an `InputText` component onto the page.

Because you chose to use automatic component binding, each component dropped on the page will automatically be bound to a property on the backing bean.

Component properties, such as `Label`, `AutoSubmit`, and `Disabled`, are set for you. You can change these as needed. Refer to the JDeveloper online help for more information about ADF Faces properties.

4. By default, the component will have an ID based on the component type, such as `inputText1`. Change this ID to be the same as the corresponding element name from the associated WSDL.

For example, [Example 9–11](#) shows the elements for a customer in the `CustomerService` WSDL:

Example 9–11 Element Definitions in the WSDL

```
<complexType name="Customer">
  <sequence>
    <element name="password" type="string" nillable="true" />
    <element name="lname" type="string" nillable="true" />
    <element name="phonenumber" type="string" nillable="true" />
    <element name="fname" type="string" nillable="true" />
    <element name="addressList" type="ns1:list" nillable="true" />
    <element name="creditcardtype" type="string" nillable="true" />
    <element name="email" type="string" nillable="true" />
    <element name="status" type="string" nillable="true" />
    <element name="creditcardnumber" type="string" nillable="true" />
    <element name="custid" type="string" nillable="true" />
  </sequence>
</complexType>
```

To create an `inputText` component for the `lname` element, you would change the ID of that component to `lname`.

5. Add code to the corresponding backing bean to collect the data and invoke the web service.

Since there are already getter and setter methods for the input components on the page, you only need to write code that creates a new object using that data.

[Example 9–12](#) shows the code to create customer and address objects.

The web service is accessed using the client generated when creating the proxy (see [Section 5.3.1, "How to Create a Web Service Proxy"](#)). Note that this method first sets the user as an authorized user, then checks to make sure that the data entered in the password and password check components is the same, and then creates a customer by accessing the web service.

Tip: You will need to import any classes used from the proxy. For example, in [Example 9–12](#), you would need to import the `Customer` and `Address` classes.

Example 9–12 Code to Create a Customer and Invoke the CustomerService Web Service

```
public void register_action(ActionEvent ae) {
    String AUTH_USER = "Authorized_User";
    FacesContext ctx = FacesContext.getCurrentInstance();
    Customer newCust = new Customer();
    if (password.getValue().toString().equals(password
        _chk.getValue().toString())) {

        // Call Web service to register new customer
        try {
            oracle.soademo.view.services.CustomerServiceClient myPort = new
                oracle.soademo.view.services.CustomerServiceClient();
            System.out.println("calling " + myPort.getEndpoint());

            // Adding new customer info
            Address addr = new Address();

            newCust.setFname(fname.getValue().toString());
            newCust.setLname(lname.getValue().toString());
            newCust.setEmail(email.getValue().toString());
            newCust.setPhonenumber(phone.getValue().toString());
            newCust.setPassword(password.getValue().toString());

            addr.setStreet(street.getValue().toString());
            addr.setCity(city.getValue().toString());
            addr.setState(state.getValue().toString());
            addr.setZip(zip.getValue().toString());

            List addrList = new ArrayList();
            addrList.add(addr);
            newCust.setAddressList(addrList);

            // Call Customer WS to add customer -
                returns customer id of new customer (newCustId)
            newCustId = myPort.addNewCustomer(newCust);

            // Retrieve complete customer object using WS

            newCust = myPort.findCustomerById(newCustId);

            // Generate successful registration message..
            FacesContext.getCurrentInstance().addMessage(null, new
                FacesMessage("Registration Successful!"));

        } catch (Exception ex) {
            FacesMessage msg = new FacesMessage("Registration Failed!");
            msg.setSeverity(msg.SEVERITY_ERROR);
            FacesContext.getCurrentInstance().addMessage(null, msg);
            FacesContext.getCurrentInstance().addMessage(null, new
                FacesMessage(ex.getMessage()));
            ex.printStackTrace();
        }
    }
}
```


6. Drag a command button from the Component Palette.

This button will be used to invoke the method just created.

7. With the command button selected, in the Property Inspector, use the dropdown menu to select the method just created.

This binds the command button to that method. When the user clicks the button, that method will be invoked, and the entered data will be sent to the web service.

9.6.4 What Happens When You Invoke a Service From a Web Page

When you drop components onto a JSF page and choose to use automatic component binding, JDeveloper does the following:

- If you elect to have JDeveloper create a backing bean, creates a JavaBean using the same name as the JSP, and places it in the `view.backing` package.
- Creates a managed bean entry in the `faces-config.xml` file for the backing bean. By default, the managed bean name is `backing_<page_name>` and the bean uses the request scope.
- On the newly created or selected bean, adds a property and accessor methods for each component tag you place on the JSP.
- Binds the component tag to that property using an EL expression as the value for its binding attribute.

[Example 9–13](#) shows the code created on a backing bean when you add two input components, one for first name and one for last name.

Example 9–13 Register Backing Bean

```
public class Register {

    private HtmlHtml html1;
    private HtmlHead head1;
    private HtmlBody body1;
    private HtmlForm form1;
    private CoreInputText fname;
    private CoreInputText lname;

    public void setHtml1(HtmlHtml html1) {
        this.html1 = html1;
    }

    public HtmlHtml getHtml1() {
        return html1;
    }

    public void setHead1(HtmlHead head1) {
        this.head1 = head1;
    }

    public HtmlHead getHead1() {
        return head1;
    }

    public void setBody1(HtmlBody body1) {
        this.body1 = body1;
    }
}
```

```
public HtmlBody getBody1() {
    return body1;
}

public void setForm1(HtmlForm form1) {
    this.form1 = form1;
}

public HtmlForm getForm1() {
    return form1;
}

public void setFname(CoreInputText inputText1) {
    this.fname = inputText1;
}

public CoreInputText getFname() {
    return fname;
}

public void setLname(CoreInputText inputText1) {
    this.lname = inputText1;
}

public CoreInputText getLname() {
    return lname;
}
}
```

[Example 9–14](#) shows the code on the JSF page binding the components to the Register backing bean properties.

Example 9–14 JSF Code Using Automatic Component Binding

```
<afh:body binding="#{backing_Register.body1}" id="body1">
  <h:form binding="#{backing_Register.form1}" id="form1">
    <af:inputText label="First Name" binding="#{backing_Register.fname}"
      id="fname"/>
    <af:inputText label="Label 2" binding="#{backing_Register.lname}"
      id="lname"/>
  </h:form>
</afh:body>
```

A command button can use either its `Action` or `ActionListener` attribute to bind to a method. The `actionListener` attribute was used instead of the `action` attribute in this case, because the navigation capabilities provided by the `action` attribute are not needed, the user remains on the registration page. When you bind the command component to the method on the backing bean, the method is invoked when the user clicks the button.

When creating a method to invoke a web service, you can invoke the web service using the client class automatically generated when creating the proxy. You first call the service, then call the needed operations on the service.

For example, the `register_action` method on the `Register.java` backing bean first calls the web service using the client class, as shown in [Example 9–15](#).

Example 9-15 Calling the CustomerService Web Service Using the Client Class

```
oracle.soademo.view.services.CustomerServiceClient myPort = new
    oracle.soademo.view.services.CustomerServiceClient();
```

The method then creates the customer by taking the value of each component, converting it to a `String`, and setting that as the value for the corresponding property on the `Customer` object, as shown in [Example 9-16](#).

Example 9-16 Creating the Customer

```
Customer newCust = new Customer();
. . .
    newCust.setFname(fname.getValue().toString());
    newCust.setLname(lname.getValue().toString());
. . .
```

Once the customer is created, the method calls the `addNewCustomer` operation on the web service, passing in the created customer, as shown in [Example 9-17](#).

Example 9-17 Calling an Operation on the Web Service

```
myPort.addNewCustomer(newCust)
```

9.7 Testing the Web Application

Procedures for testing the web application vary depending on whether or not you elect to use ADF data binding in the application. If you do not use ADF data binding, your testing and debugging scenarios can follow any generic web testing and debugging procedures.

To specifically test and debug a web service invocation, you can use the client class generated when you created the proxy to the web service. [Example 9-18](#) shows the code you might use to test the web service by finding a customer whose ID is 1.

Example 9-18 Testing the Web Service Using the Client Class

```
public static void main(String[] args) {
    try {
        model.proxy.CustomerServiceClient myPort = new
            model.proxy.CustomerServiceClient();
        System.out.println("calling " + myPort.getEndpoint());
        // Add your own code here
        myPort.findCustomerById("1");
    } catch (Exception ex) {
        ex.printStackTrace();
    }
}
```

You can also test the web service using JDeveloper's HTTP Analyzer. For more information, see [Section 5.4, "Debugging, Testing and Analyzing Web Services in JDeveloper"](#).

Most failures in the web application's interaction with Oracle ADF result from simple and easy-to-fix errors in the declarative information that the application defines or in the EL expressions that access the runtime objects of the page's Oracle ADF binding container. Refer to *Oracle Application Development Framework Developer's Guide* for more information regarding troubleshooting and debugging ADF applications.

Securing the System

This chapter explains how to secure deployed services with Oracle Web Services Manager. Using the Credit Validation Service of the SOA Order Booking Application as an example, it describes the facilities provided by Oracle Web Services Manager to protect web services and process flows in a service-oriented environment, without having to modify either client applications or web services.

Note: This chapter contains specific examples of securing web services with Oracle Web Services Manager. For comprehensive details about product features and capabilities, see the *Oracle Web Services Manager Administrator's Guide*.

Note: Enterprise level applications (for example, a J2EE application that uses Oracle ADF) may require application-level security. For example, a typical J2EE application may have a login page, and certain areas of the application should be restricted based on a user's role within an organization. This chapter focuses only on securing services within an SOA system. For more information about using application-level security, refer to the *Oracle Containers for J2EE Security Guide*. For more information about using security in an Oracle ADF application, refer to *Oracle Application Development Framework Developer's Guide*.

This chapter includes the following sections:

- [Section 10.1, "Introduction to Oracle Web Services Manager"](#)
- [Section 10.2, "Securing Web Services Using Oracle Web Services Manager"](#)
- [Section 10.3, "Authenticating Users with an Oracle Web Services Manager Server Agent"](#)
- [Section 10.4, "Encryption with an Oracle Web Services Manager Gateway"](#)

10.1 Introduction to Oracle Web Services Manager

Oracle Web Services Manager (Oracle WSM) provides a policy enforcement framework to manage, secure, and monitor web services consistently and flexibly across organizational boundaries. It enables organizations to employ a common security infrastructure across all their web services applications, providing the operational visibility and control, including service level agreement (SLA) management capabilities, required to deploy web services in production. Oracle Web Services Manager achieves this through policies, which are a set of tasks (such as logging and authentication) that are performed at specific policy enforcement points, as service requests and responses between a service client and a service provider are processed.

Oracle WSM secures your services environment with these key components:

- Oracle WSM Policy Manager

The Oracle WSM Policy Manager allows you to define policies that reflect operational best practices and requirements. It includes a browser-based tool for creating and maintaining security and management policies for web services and business processes, using prebuilt or custom policy steps. Examples of actions performed by policy steps are:

- Performing an authorization
- Logging an audit record
- Performing an LDAP authentication
- Decrypting an XML payload

With the Oracle WSM Policy Manager, you can configure and manage best practice policies, and ensure that these policies are enforced regardless of the details of the service or its implementation.

- Oracle WSM Gateways

Gateways provide a non-intrusive mechanism for policy enforcement.

Gateways provide several key features:

- Gateways operate independently of the protected services, acting as a proxy to service clients.
- Gateways virtualize the underlying web service, so that the address details of the service are not visible to clients.
- Gateways can perform routing based on message content or attachments.
- Gateways can perform message transforms from one format or protocol to another; for example, from XML-over-HTTP to JMS.

- Oracle WSM Agents

Also serving as policy enforcement points, agents are lightweight components that run in the same "container" or application server environment as the web service. Oracle Web Services Manager provides two types of agents, a client agent and a server agent:

- A client agent secures web service clients. It is embedded into web service clients, fetches the policies from Oracle WSM Policy Manager, and does not need modification of the deployment EAR or WAR files.

- A server agent secures web services. It is embedded into the web service, and it also reads its policies from the Oracle WSM Policy Manager. It differs from client agents in that it does not require modification and redeployment of EAR or WAR files.

Note: While agents can support a majority of Oracle Web Services Manager's prebuilt policy steps, they do not support message routing or transformation.

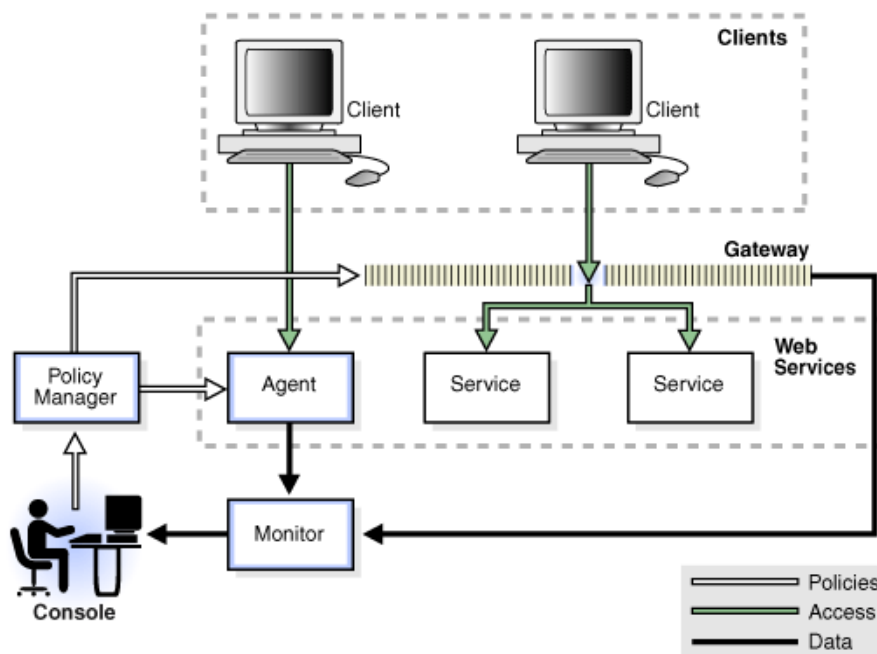
See Also: [Section 10.2.1, "When to Use Agents or Gateways"](#)

■ Oracle WSM Monitor

The Oracle WSM Monitor component manages collection and aggregation of web services traffic data and provides alerts and notifications. As the gateways and agents enforce policies on incoming and outgoing messages, they collect statistics about response times, exceptions, and so on. These statistics are sent in real time to the Oracle Web Services Manager Monitor, which is a web-based dashboard for monitoring service-level agreements (SLAs), service availability, and service responsiveness. The monitor can alert administrators when boundary conditions are met, and it can also automatically communicate with the Oracle WSM Policy Manager to activate new policies under certain conditions.

Figure 10–1 shows the key components of Oracle Web Services Manager.

Figure 10–1 Components of Oracle Web Services Manager



This figure shows how the Oracle WSM Policy Manager manages and monitors client access to web services, enforcing the operational policies you have configured for the agents and gateways. Oracle Enterprise Manager 10g Web Services Manager Control, which is the user interface component of the Policy Manager, is used to configure policies and monitor web services traffic.

Read this chapter to understand:

- How Oracle WSM uses policies, and the purpose of policy enforcement points
- How to secure web services by defining agents, and registering the services to be protected by the agents
- How to use an Oracle WSM Client Agent to authenticate user credentials
- How to use an Oracle WSM Gateway to encrypt data exchanged with a web service

10.2 Securing Web Services Using Oracle Web Services Manager

This section explains Oracle Web Services Manager security policies and policy pipelines, and provides a high-level view of the implementation steps.

10.2.1 When to Use Agents or Gateways

Oracle Web Services Manager Agents are of interest to sites that wish to implement end-to-end security. This is designed to assure that the business process is secured from the very beginning to the very end of its execution. Agents are also useful in settings where the service endpoint is already known to clients and it would not be desirable to change the endpoint.

Gateways, on the other hand, are used when there is a need to deploy a security choke point. This is similar to a firewall, where security policies can be employed in a central location. A gateway can also perform functions that an agent cannot do, such as message routing, transformations, and failover.

Note: Agents can be used with gateways. The use of one does not preclude the use of the other.

10.2.2 How Oracle Web Services Manager Uses Policies

Policies are used to manage and secure web services: they include tasks such as authentication, authorization, encryption, decryption, protocol transformation, and so on.

As mentioned earlier, policies can be deployed at two kinds of policy enforcement points, gateways and agents.

An administrator can assemble policies from:

- Predefined policy steps that ship with Oracle Web Services Manager
- Custom policy steps

Steps are assigned to any of four policy pipelines, depending on where in the message stream the step is to be applied:

- Pre-Request Pipelines contain policy steps to be enforced when preprocessing incoming web service requests.
- Request Pipelines contain policy steps to be enforced when processing incoming requests.
- Response Pipelines contain policy steps to be enforced when processing outgoing requests.

- Post-Response Pipelines contain policy steps to be enforced when processing outgoing requests, after any policy steps in the response pipeline have been processed.

About Pre-Request and Post-Response Pipelines

Oracle Web Services Manager provides two sets of pipelines on both incoming and outgoing streams to allow flexibility in managing the services.

Pre-request and post-response pipelines can be used to set company and department-wide policies. For example, the pre-request pipeline is configured when preprocessing incoming web service requests.

The request and response pipelines, on the other hand, are used to provide additional policy enforcement steps locally for specific services. Administrators of those services can make use of the pipelines to implement additional policy steps.

10.2.3 Security Architecture

In the SOA Order Booking Application, a customer signs on to the application through a web interface and orders a product. When the customer clicks the **Place Order** button, the OrderBookingESB flow (developed with the Oracle Enterprise Service Bus framework) is invoked. This flow, in turn, routes information from the web client and invokes the SOAOrderBooking BPEL flow.

The SOAOrderBooking BPEL flow then handles the order process; after the order is inserted into a database table, the customer data is sent to the Credit Validation Service for credit card verification.

Some security concerns must be addressed in this application flow:

- Authentication of incoming credentials received by the Oracle BPEL Process Manager

An Oracle WSM Server Agent can be deployed to BPEL partner services in order to enforce security on these services. For example, one could require authentication to a service using a WS-Security username token.

[Section 10.3, "Authenticating Users with an Oracle Web Services Manager Server Agent"](#) explains how to implement authentication between Oracle BPEL Process Manager and an agent-secured partner service.

- Encryption of customer credit card data sent over the network to the Credit Validation Service

An Oracle WSM gateway can be configured to protect the customer data exchange with the Credit Validation Service by means of data encryption and digital signatures.

[Section 10.4, "Encryption with an Oracle Web Services Manager Gateway"](#) describes the procedure for implementing authentication.

Role of Oracle Web Services Manager in the Demo Application

Web services security is not configured "out of the box" for the services participating in the SOA Order Booking Application. This chapter provides examples to illustrate how the services can be secured with Oracle Web Services Manager without any programmatic effort. Since Oracle Web Services Manager is an integrated component of the Oracle SOA Suite, no additional installation tasks are required to follow and replicate the examples presented here.

Although most of the configuration is accomplished using the UI tools in the Oracle SOA Suite, a few steps require the use of command-line tools as explained in the text of the examples.

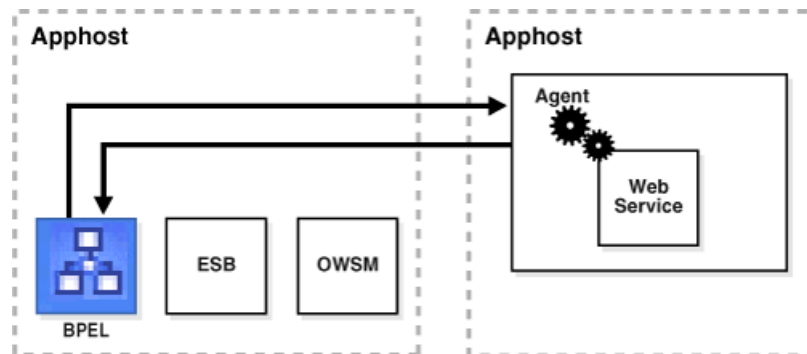
10.3 Authenticating Users with an Oracle Web Services Manager Server Agent

Note: This section illustrates a specific use case for providing web services security with Oracle Web Services Manager. It does not provide comprehensive details about Oracle WSM product features and capabilities.

In the SOA Order Booking Application, an Oracle Web Services Manager server agent can be deployed to enforce authentication by verifying the customer's user name and password against an LDAP repository, with Identity Management infrastructure like Oracle Access Manager or SiteMinder, or by means of a simple file-based repository.

Figure 10–2 shows how an Oracle Web Services Manager agent can be utilized for this purpose in the demo application. The agent intercepts requests from Oracle BPEL Process Manager clients to the web service. It extracts the client credentials and verifies the customer's user name against data contained in an operating system file.

Figure 10–2 Authentication with Oracle Web Services Manager



Note: The use of an agent illustrates just one approach for performing this task with Oracle Web Services Manager. For a comparison of agents and gateways, see [Section 10.2.1, "When to Use Agents or Gateways"](#).

See Also: The *Oracle Web Services Manager Administrator's Guide* for details about Oracle Web Services Manager's authentication capabilities

The general steps to provide user name authentication are:

1. Register an Oracle Web Services Manager server agent.
2. Define the policy set for the server agent.

The policy set dictates the specific security and management steps that Oracle Web Services Manager will enforce.

3. For test environments, set up a file-based repository of user credentials.
The example in this use case utilizes a file repository.
4. Install the server agent as a J2EE agent.
5. Configure the Credit Validation Service web service agent in Application Server Control Console.
6. Test the authentication.
7. Configure the Oracle BPEL Process Manager process to send the username token, which is validated by the Oracle Web Services Manager server agent.

10.3.1 How to Register a Server Agent

You use the Web Services Manager Control Console to define Oracle Web Services Manager components such as server agents. The server agent acts as an enforcement point for security policies.

To configure a server agent component in Oracle Web Services Manager:

1. Point your browser to the Web Services Manager Control Console and log in using your single sign-on user name and password.

The Web Services Manager Control Console is accessed with a URL of the form:

`http://<hostname>:port_number/ccore`

For example:

`http://itapps.globalcompany.com:8888/ccore`

2. Click **Add New Component**.
3. On the Add New Component page, define:

- Component name
- Component type
- Container type

[Figure 10–3](#) shows how a server agent named the Authentication Agent is defined:

Figure 10–3 Adding a Server Agent

Enforcement Points [Help](#)

Add New Component:

Basic Parameters	
Component Name:	Authentication Agent
Component Type:	Server Agent
Container Type:	OC4J

Component Groups:	
Modify privileges su1-grp da1-grp	Add Groups with Modify privileges ca1-grp ca2-grp
<< Add	
View privileges	Add Groups with View privileges cs1-grp cs2-grp
Remove >>	

Register Cancel

Note that the component type is Server Agent.

- Click **Register**. A confirmation page appears, as shown in [Figure 10–4](#):

Figure 10–4 Oracle WSM Component ID Confirmation**Enforcement Points**

Component is added successfully. Id of the new Component is "C0003002".

Ok

- Make a note of this unique component ID for later use, and click **OK**.

The new agent appears in the list of components.

10.3.2 What Happens When You Register an Agent

When you register an agent, Oracle WSM adds it to the server system registry. It also assigns a unique component ID to the agent, identifying it for all actions across your deployment.

10.3.3 How to Define the Policy Set for the Server Agent

A policy set consists of the explicit security and management policies that are to be implemented at a server agent. Oracle Web Services Manager offers a large selection of policy steps that can be combined as appropriate for specific security tasks.

For example, the Credit Validation Service can be protected by defining a policy set for the server agent consisting of: Oracle Web Services Manager

- A policy step to extract credentials contained in the SOAP security headers
- A policy step to authenticate the credentials

These policy steps must be defined at the agent's request pipeline.

See Also: ["How Oracle Web Services Manager Uses Policies"](#) on page 10-4

You use the Web Services Manager Control Console to define the desired policies for the agent. After selecting the agent of interest, you can choose whether to update the default policy or add a new policy. Next, define one or more policy steps for the selected policy.

To define the policy set for the server agent:

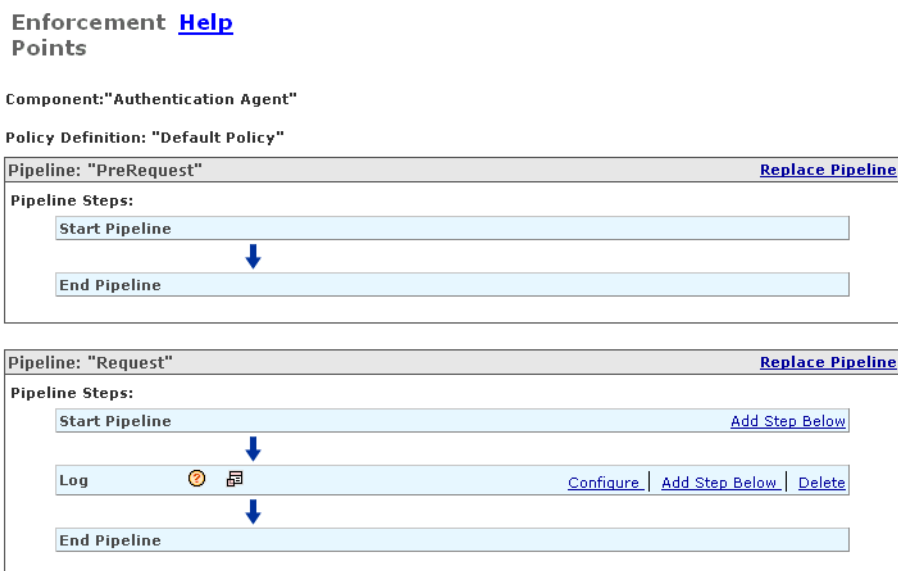
1. At the Web Services Manager Control Console, click **Policy Management**, then click **Manage Policies**.
2. In the component list, select the agent for which the policy set is to be defined, and click **Policies**.

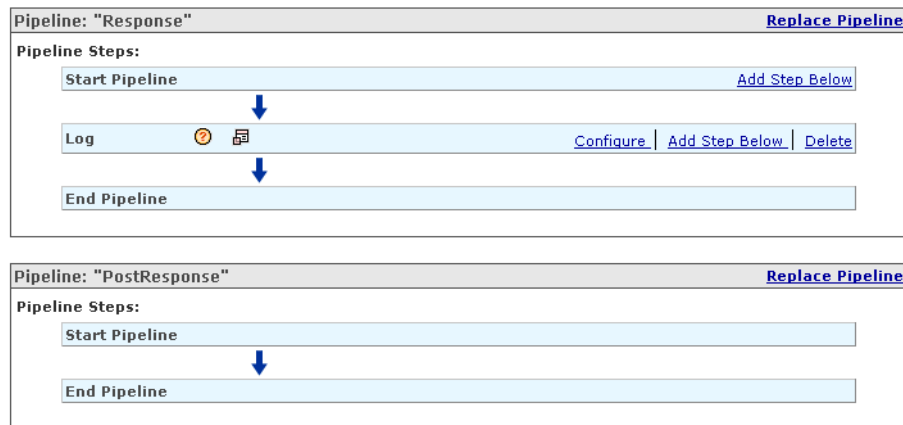
The policy set for the agent is displayed.

Oracle Web Services Manager provides a default policy for each agent or gateway component, and you have the ability to add additional policies to implement additional policy controls.

3. Click the **Edit** icon for the Default Policy. The Policy Definition page appears. [Figure 10-5](#) shows the default policy pipeline for Authentication Agent:

Figure 10-5 Policy Pipeline for the Default Policy



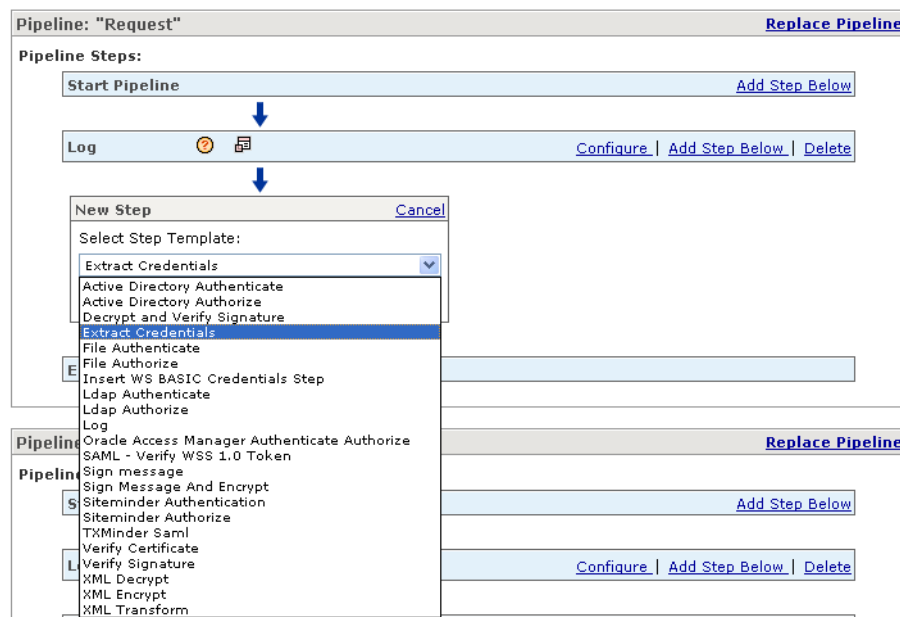


The Web Services Manager Control Console displays the current contents of the policy, namely the steps in the Pre-Request, Request, Response, and Post-Response pipelines. A log step is enabled for the Request and Response pipelines by default.

4. In the request pipeline, implement credential authentication following the Log policy step. Click **Add Step Below**.

A box appears at the appropriate location, allowing you to select a policy step as shown in Figure 10-6:

Figure 10-6 Inserting a Policy Step at the Request Pipeline



5. Select **Extract Credentials** from the list, and click **OK**.

The Policy Definition page reappears, showing the newly added Extract Credentials step.

6. Click **Configure** to define the properties for the Extract Credentials step, as illustrated in Figure 10-7:

Figure 10–7 Defining Properties for the Extract Credentials Step

Configure Step [Help](#)

Pipeline: "Request"

Configure pipeline step

Pipeline Step Name: Extract Credentials

Extract Credentials		Environmental Properties	
Basic Properties		Type	Default
Enabled (*)	boolean	true	☒ true ☐ false
Extract Credentials Properties		Type	Default
Credentials location (*)	string	HTTP	WS-BASIC
Namespaces	string		
UserID xpath	string		
Password xpath	string		

↓ ↓

Faults and Fault Handlers

Fault Code: [http://schemas.oblix.com/ws/2003/08/Faults:AuthenticationFault](#) [Add Handler](#)

Ok Cancel

On the Configure Step page shown in this figure:

- The Enabled property is set to `true` by default.
- The credentials location is set to `WS-BASIC`.

The `WS-BASIC` location indicates that credentials are to be extracted from the standard UsernameToken as specified in the WS-I Basic Security Profile.

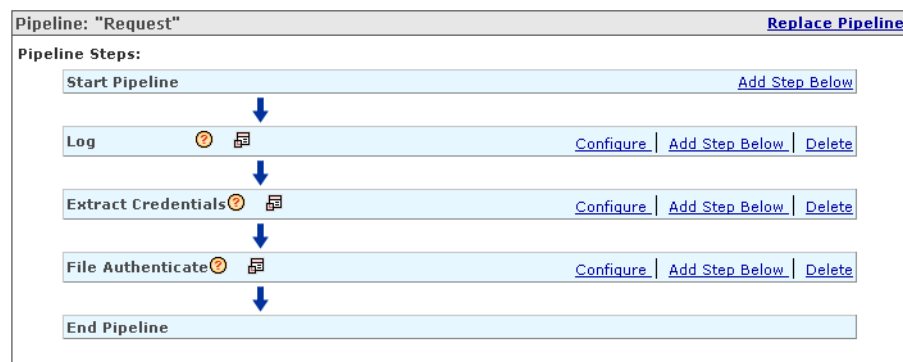
7. Click OK.

The Policy Definition page reappears.

8. To add the second step for credential authentication, in the Extract Credentials row, click **Add Step Below.**

9. In the New Step box, select the step template named File Authenticate, and click **OK.**

The request pipeline steps are displayed, and File Authenticate appears as the last step in the pipeline as shown in [Figure 10–8](#):

Figure 10–8 Request Pipeline for the Server Agent

10. To define properties for the new step, click **Configure** in the File Authenticate row. The Configure Step page appears, as shown in Figure 10–9:

Figure 10–9 Defining Properties for the File Authenticate Step

Configure Step [Help](#)

Pipeline: "Request"

Configure pipeline step

Pipeline Step Name: File Authenticate

File Authenticate		Environmental Properties	
Basic Properties	Type	Default	Value
Enabled (*)	boolean	true	☒ true ☐ false
Authentication Properties	Type	Default	Value
Passwd file location (*)	string	.htpasswd	C:\admin\htpasswd
.htpasswd file format (*)	string	mixed	md5

Faults and Fault Handlers

Fault Code: http://schemas.oblix.com/ws/2003/08/Faults:AuthenticationFault [Add Handler](#)

Ok Cancel

11. Define these properties for the step:
- Specify C:\admin\htpasswd as the Passwd (password) file location.
 - Select md5 as the .htpasswd file format.

Note: The step is enabled by default.

12. Click **OK** to complete the step definition. The Policy Definition page reappears.
13. Click **Next** to return to the policy page.
14. Click **Save** to save the step definitions.
15. Click **Commit** on the Policy Set page.

10.3.4 What Happens When You Define the Policy Set for the Agent

After you define one or more policies and their respective policy steps, they do not take effect immediately. You can continue to update or amend the policy configuration for the agent as needed, for example, by adding more policy steps to a policy pipeline. Once you are satisfied with the step definitions that have been saved, the agent's policy set is displayed as in [Figure 10–10](#):

Figure 10–10 Committing Policy Changes

Enforcement Points [Help](#)

Name : Authentication Agent View Versions [Version](#)
 Type : ServerAgent Save Policy [Save](#)
 Commit Policy [Commit](#)

Policy Set for Component: "C0003002" [Add New Policy](#)

Policy Name	View Details	Edit	Delete
Default Policy			

[Edit Mappings](#)

URL Pattern	Policy Name
*	Default Policy

"Commit Policy" appears in red on the page, and the Oracle WSM Policy Manager is not updated with the new information until you click **Commit**. Once this is done, the component's policy set becomes effective, and remains in effect until subsequent changes are made and committed.

10.3.5 How to Set Up a File with User Credentials

In the previous step, [Section 10.3.3, "How to Define the Policy Set for the Server Agent"](#), a File Authenticate step was configured. As noted there, file-based authentication is typically used in test environments.

The format of the credential file is the same as the `.htpasswd` file format used by the Apache Web server. The password can be encoded in four forms: MD5, SHA1, plain text, or some mix of the three forms.

To set up a file with user credentials:

1. Use a text editor such as Notepad or vi to create the file. Add a single user name entry to the file, for example:

```
bill:
```

Save and close the file.

2. Use the Oracle WSM command-line tool, found at `ORACLE_HOME\owsm\bin\wsadmin`, to complete the file.

The command syntax is:

```
wsadmin md5encode user_name password htpasswdfile
```

For example:

```
C:\OracleAS_1\wsadmin md5encode bill billspwd .htpasswd
```

See the *Oracle Web Services Manager Deployment Guide* for details.

10.3.6 How to Install the Server Agent

The server agent must be installed as a J2EE agent. Oracle Web Services Manager provides the `wsadmin` command-line Oracle Web Services Manager tool Oracle Web Services Manager for a number of administrative tasks, including agent registration.

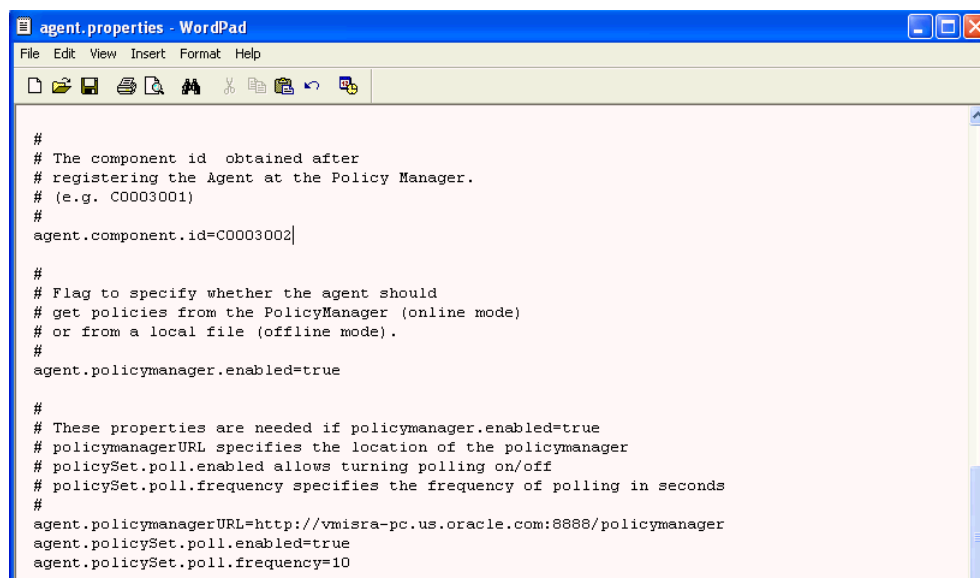
To install the server agent as a J2EE agent:

1. At a Oracle Web Services Manager command prompt, navigate to the `bin` directory of the Oracle Web Services Manager component within your Oracle SOA installation. For example:

```
ORACLE_HOME\owsm\bin
```

2. Using a text editor, edit the `agent.properties` file.
3. For the `agent.component.id`, insert the Component ID of the server agent that was registered. [Figure 10–11](#) provides an example:

Figure 10–11 Editing the `agent.properties` File



Save the file.

4. At the command prompt, run the `installAgent` command:

```
ORACLE_HOME\owsm\bin\wsadmin installAgent Oracle-AS-password
```

where *Oracle-AS-password* is the OC4J administrator password.

The command is executed to build and install the J2EE server agent, as shown in [Figure 10–12](#):

Figure 10–12 Installing the Server Agent

```

C:\0003002\config\serveragent
[copy] Copying 1 file to C:\product\10.1.3.1\OracleAS_1\owsm\config\interc
ptors\0003002
[copy] Copying 7 files to C:\product\10.1.3.1\OracleAS_1\owsm\config\interc
eptors\0003002\config
[copy] Copying 1 file to C:\product\10.1.3.1\OracleAS_1\owsm\config\interc
ptors\0003002\config\serveragent
[copy] Copying 1 file to C:\product\10.1.3.1\OracleAS_1\owsm\config\interc
ptors\0003002\config\serveragent

Interceptor.copyLibs:
[echo] copying libs now

agent.cleanup:
[delete] Deleting directory C:\DOCUME~1\UMISRA~1\ST~1\LOCALS~1\Temp\coresv.aga
ntinstaller

BUILD SUCCESSFUL
Total time: 13 seconds

```

10.3.7 How to Configure the Web Services Agent in Application Server Control

You must enable the Oracle Web Services Manager agent, Oracle Web Services Manager and associate it with the Oracle WSM Server Agent.

To Configure the web services Agent in Enterprise Manager:

1. Begin at the Oracle Enterprise Manager 10g Application Server Control Console home page.
Under **Members**, click the home link.
2. Click **Web Services**. The list of web services is displayed.
3. Click the link for the `ValidateCreditCardServiceSoapHttp` web service. The main page for the service is displayed, as illustrated in [Figure 10–13](#):

Figure 10–13 Web Service Main Page

Web Service: `ValidateCreditCardServiceSoapHttp` Page Refreshed Jun 15, 2006 3:09:07 PM

[Home](#) [Operations](#) [Administration](#)

General		Performance	
Application	Up Disable	Response Time (milliseconds)	Unavailable
Status	Enabled	Invocations	Unavailable
Port Enabled	Unavailable	Faults	Unavailable
Start Time	java-class		
Type	document		
Style			
Service Name	CreditService		
Module	WebServices		
Application	SOADEMO-CREDITSERVICE-CreditService-WS		

Related Links

[Test Service](#) [View Diagnostic Log Messages](#)

[Home](#) [Operations](#) [Administration](#)

4. Click the **Administration** link to display the management features available for this web service, as shown in [Figure 10–14](#):

Figure 10–14 Web Service Administration Page

Web Service: ValidateCreditCardServiceSoapHttp Page Refreshed Jun 15, 2006 3:14:06 PM PDT

[Home](#) [Operations](#) [Administration](#)

This table lists the management features available. You can customize the configuration of these management features and selectively enable/disable features. The customized configuration for a given management feature will only affect the runtime when the feature is enabled.

Feature	Edit Configuration	Enabled	Description
Auditing			Record message contents for purpose of auditing and/or non-repudiation.
Reliability			Configure port level reliability characteristics for web service messages.
Security			Configure security features to be applied to web service requests and responses.
Tracing			Extract contents of messages and store them.
Web Services Agent			Configure the Web Services agent for Oracle Web Services Manager.

- To enable the web services Agent feature, click **Enable/Disable Features**. The list of available features appears.
- Move Web Services Agent to the Enabled Features list, as shown in [Figure 10–15](#):

Figure 10–15 Enabling the Web Services Agent

Enable/Disable Features Cancel OK

Page Refreshed Jun 15, 2006 3:17:35 PM PDT

Select the management features that you would like to enable. Note that you can change the configuration for a management feature when the feature is disabled, but to have the configuration changes affect the runtime, the management feature has to be enabled.

Available Features

- Reliability
- Security
- Tracing
- Auditing

>

Move

>>

Move All

<

Remove

<<

Remove All

Enabled Features

- Web Services Agent

Cancel OK

- Click **OK**. The Web Service:ValidateCreditCardServiceSoapHttp page reappears, with the web services Agent enabled
- Note the confirmation message at the top of the page.
- The final configuration step involves associating the component ID for the Oracle Web Services Manager Server Agent with this web service agent.
- On the ValidateCreditCardServiceSoapHttp web service management features page, shown in [Figure 10–14](#), click the **Edit Configuration** icon for the Web Services Agent.
- The Edit Web Services Agent Configuration page appears.
- In the Configuration Directory box, enter the component ID of the Oracle Web Services Manager Server Agent, as shown in [Figure 10–16](#):

Figure 10–16 Editing Web Services Agent Configuration

Edit Web Services Agent Configuration Cancel OK

Page Refreshed Jun 15, 2006 4:29:01 PM PDT

Enter or select the configuration directory that contains the files for the Web Services Agent registered for this Web Service. Typically, this is the same as the component identifier generated by Oracle Web Services Manager for this server agent.

• Configuration Directory

Multiple webservices in a webmodule should share the same configuration directory.

Cancel OK

10. Click **OK**. A confirmation is displayed on the web service management features page.

10.3.8 What Happens When You Configure the Web Services Agent in Application Server Control

Configuring the web services agent in Application Server Control makes the Oracle WSM Server Agent known to the application server. The integration of Oracle WSM in the Oracle SOA Suite enables you to configure this management information simply by supplying the agent's Oracle WSM component ID. Note that the management information is completely separate from the web service, business logic, and client implementation.

10.3.9 How to Test the Authentication

The final step is to test the agent that you have configured for credential authentication.

To test the authentication:

1. Return to the Applications link of the OC4J home instance and expand the default application.
2. Click the link for the application of interest to display the application home page. In the example, this is the **SOADEMO-CREDITSERVICE-CreditService-WS** link.
3. Select the Web Services tab. The **SOADEMO-CREDITSERVICE-CreditService-WS** application contains a single web service, as shown in [Figure 10-17](#):

Figure 10-17 Available Web Services for Application

Application: SOADEMO-CREDITSERVICE-CreditService-WS Page Refreshed Jun 15, 2006 4:46:42 PM PDT

Home	Web Services	Performance	Administration

Test Service

Select	Port Name	Web Service	Application Status	Port Enabled	Start Time	Invocations	Response Time (milliseconds)	Faults
☒	ValidateCreditCardServiceSoapHttp	CreditService	↑	✓	Unavailable	Unavailable	Unavailable	Unavailable

4. Click **Test Service**. A list of URLs appears, showing the sites that can be used to test the web service. For the Credit Validation Service, there is just one URL as shown in [Figure 10-18](#):

Figure 10-18 Test Web Service Page

Test Web Service: ValidateCreditCardServiceSoapHttp Page Refreshed Jun 15, 2006 4:49:29 PM PDT

Discovered Websites

This table shows websites that may be used to test this Web Service.

URL	http://vmisra-pc:8888/CreditService/ValidateCreditCardServiceSoapHttp			Test Web Service
Select	Listener	Protocol	Host	Port
☒	J2EE Website	http	vmisra-pc	8888

TIP The URL field may be modified to test the web service using a custom URL.

5. Click **Test Web Service**. The service's test page is displayed.
6. Expand WS-Security. Input fields, such as the User Name and Password fields in the example, appear.

7. Select the **Enable** checkbox, and enter the user credentials and credit card information as shown in [Figure 10–19](#):

Figure 10–19 Entering Information for Web Service Test

CreditService endpoint

For a formal definition, please review the [Service Description](#).

Download the JavaScript Stub (BETA) for [ValidateCreditCardServiceSoapHttp](#) and see its [documentation](#).

ValidateCreditCardServiceSoapHttp

Operation: **VerifyCC** Edit As: **HTML Form**

☒ **Reliable Messaging** ☐ **Enable**

☒ **WS-Security** ☒ **Enable**

User Name: xsd:string

Password: xsd:string

CreditCard

ccType: xsd:string

ccNum: xsd:string

☐ Show Transport Info

☐ Perform stress test

Invoke

Click **Invoke**. A Test Result page appears.

8. You can validate the results by querying the Oracle Web Services Manager Oracle Web Services Manager logs. From the Web Services Manager Control Console, click **Operational Management**, then click **Overall Statistics**, and click **Message Logs**.
9. Click **Search** to display message logs, as shown in [Figure 10–20](#):

Figure 10–20 Oracle WSM Message Logs

Message Logs [Help](#)

Message Logs Search Criteria

Component: **Authentication Agent** Time Range: **Last hour**

Search

Index	Service Id	Access Time	Log Type
1	/CreditService/ValidateCreditCardServiceSoapHttp	Thursday, June 15, 2006 05:04:00 PM	Request
2	/CreditService/ValidateCreditCardServiceSoapHttp	Thursday, June 15, 2006 05:04:02 PM	Response

Note that two logs, a request log and a response log, were generated for the Credit Validation Service over the last hour.

10. Clicking the number in the Index column for the request log opens a page which displays the parameters that were passed in, as illustrated in [Figure 10–21](#):

Figure 10–21 Request Log Entry

```

- <soap:Envelope>
- <soap:Header>
  - <wsse:Security soap:mustUnderstand="1">
    - <wsse:UsernameToken>
      <wsse:Username>jstein</wsse:Username>
      <wsse:Password
        Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-
      </wsse:UsernameToken>
    </wsse:Security>
  </soap:Header>
- <soap:Body>
  - <ns1:CreditCard>
    <ccType>AMEX</ccType>
    <ccNum>12345678</ccNum>
  </ns1:CreditCard>
</soap:Body>
</soap:Envelope>

```

The response log entry shows that the credentials were validated as `true`, as in Figure 10–22:

Figure 10–22 Response Log Entry

```

- <env:Envelope>
- <env:Body>
  <ns0:valid>true</ns0:valid>
</env:Body>
</env:Envelope>

```

10.3.10 What Happens at Runtime

In the preceding example, Oracle Web Services Manager performed two key actions at runtime:

1. User credentials were extracted.

Message credentials, typically user name and password combinations, are delivered in various ways: in the transport headers, as SOAP headers, or in the XML body. The example utilized the WS-BASIC soap security header, which extracts credentials from the standard UsernameToken specified in the WS-I Basic Security Profile.

Oracle Web Services Manager provides other options for credential extraction. See the discussion of the Extract Credentials policy step in the *Oracle Web Services Manager Administrator's Guide* for more information.

2. The credentials were authenticated.

In addition to the file-based authentication in the example, Oracle Web Services Manager supports authentication against LDAP directories and security data stores. Authentication may also be presented and verified with an X.509 certificate.

Oracle Web Services Manager provides a variety of credential management, authentication, authorization, message security, and other features. For Oracle Web

Services Manager a discussion of available runtime actions, see the appendix for "Oracle WSM Policy Steps" in the *Oracle Web Services Manager Administrator's Guide*.

Error Handling

If policy execution (such as user authentication) fails, Oracle Web Services Manager returns a SOAP fault to the client. Oracle Web Services Manager

10.3.11 How to Configure Oracle BPEL Process Manager to Send a Username Token

To interact with a secured web service, the client must send SOAP messages containing valid security credentials. To achieve this, you can configure Oracle BPEL Process Manager to send username tokens that include the relevant WS-Security header tags in the `bpel.xml` file.

Note: `bpel.xml` is the BPEL metadata file which contains partner links, WSDLs, and so on.

The following portion of `bpel.xml`, shown in [Figure 10–23](#), illustrates how to enable a username token for the `CreditValidationService` partner link:

Figure 10–23 Enabling a Username Token

```
<partnerLinkBinding name="CreditValidatingService">
  <property name="wsdlLocation">CreditValidatingService.wsdl</property>
  <property name="wsseHeaders">credentials</property>
  <property name="wsseUsername">jstein</property>
  <property name="wssePassword">welcome1</property>
</partnerLinkBinding>
```

Here, credentials for the user `jstein` with password `welcome1` are set by means of the `wsseHeaders`, `wsseUsername`, and `wssePassword` deployment descriptors; these properties will be validated by the authenticating Oracle Web Services Manager Server Agent.

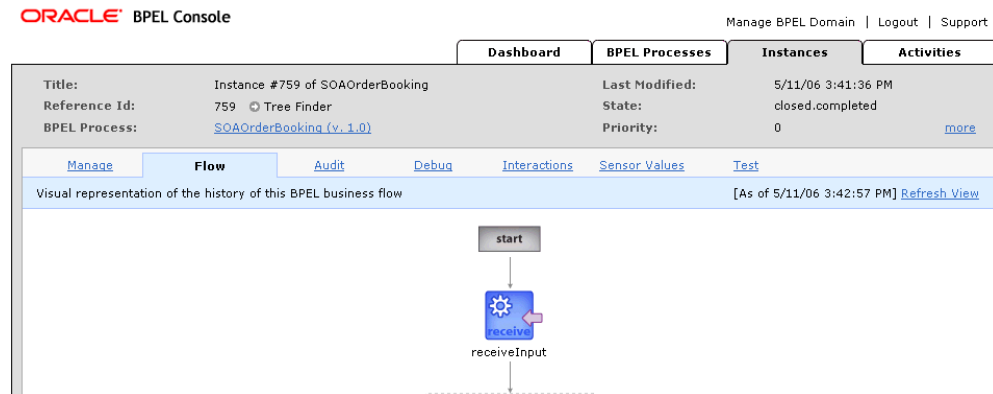
10.4 Encryption with an Oracle Web Services Manager Gateway

This section describes the procedure for encrypting data using an Oracle Web Services Manager Gateway.

In the SOA Order Booking Application, business processes handle sensitive data which is not secured, as can be verified by reviewing how data is sent within the process flow.

To review how data is sent:

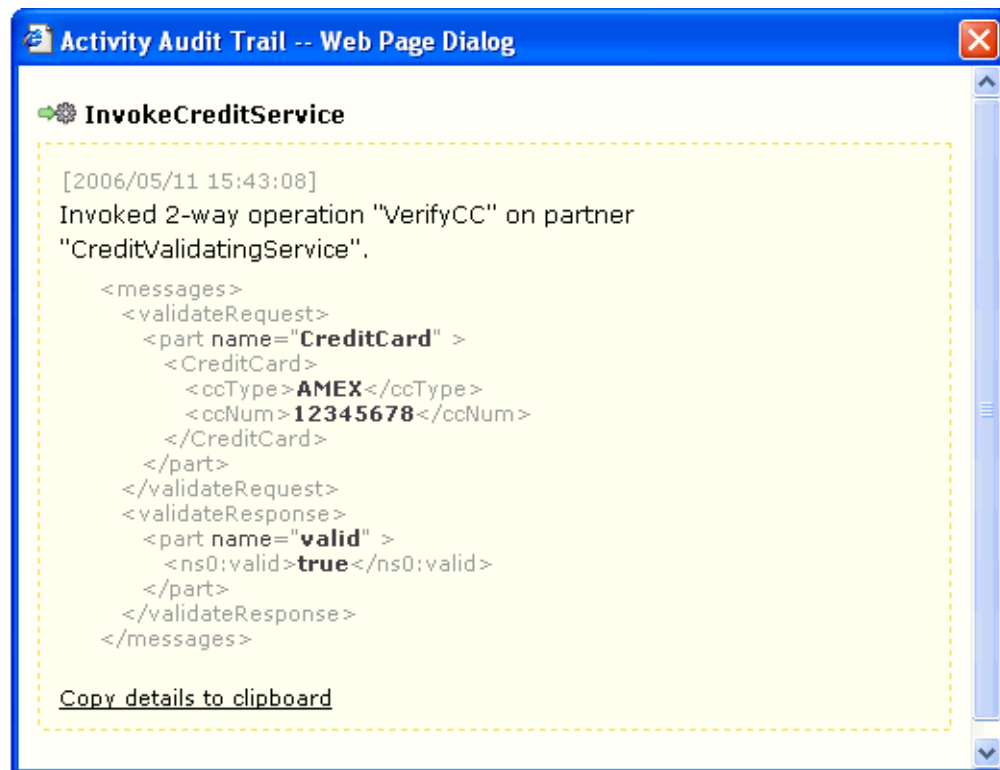
1. Using the Oracle BPEL Process Manager Console, review the SOA Order Booking Application process flow shown in [Figure 10-24](#):

Figure 10-24 The SOA Order Booking Application Process Flow

2. Scroll down to review the CreditService flow, as in [Figure 10-25](#):

Figure 10-25 Web Service Flow

3. Click on InvokeCreditService for message details, as shown in [Figure 10-26](#):

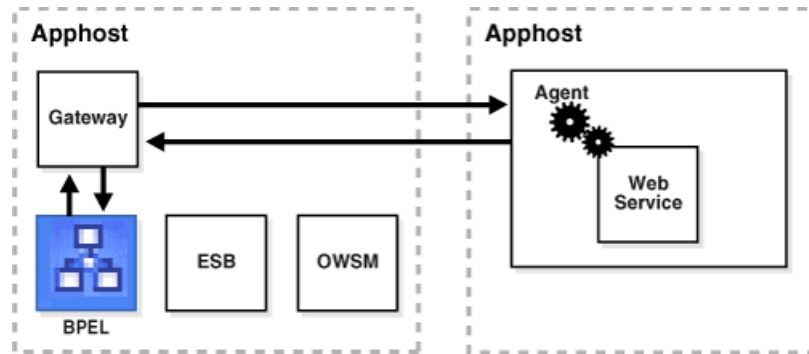
Figure 10–26 Service Invocation Details

Oracle BPEL Process Manager sends the user's credit card details, in clear text, to the Credit Validation Service for validation.

To avoid sending this data in clear text, an Oracle Web Services Manager gateway can be set up and used to encrypt data being sent from the Oracle BPEL Process Manager process to the Credit Validation Service. The Oracle Web Services Manager agent will authenticate the user, then decrypt and hand off the message to the partner link.

Note: The use of a gateway illustrates just one approach for performing this task with Oracle Web Services Manager. For a comparison of agents and gateways, see [Section 10.2.1, "When to Use Agents or Gateways"](#).

Figure 10–27 shows how an Oracle Web Services Manager Gateway is utilized for this purpose in the SOA Order Booking Application. The gateway encrypts data that clients send to the web service.

Figure 10–27 Encryption with Oracle Web Services Manager

See Also: ["Security Architecture"](#) on page 10-5 for details of the process flow

The steps to accomplish Oracle Web Services Manager encryption are as follows:

1. Register the Oracle Web Services Manager Gateway.
2. Locate the WSDL for the web service.
3. Register a policy for the services protected by the gateway.
4. Redirect web service clients to the gateway.
5. Create the certificate keystores.
6. Define the encryption and decryption policy steps.
7. Test the message encryption.

10.4.1 How to Register the Gateway

Oracle Web Services Manager Gateways serve as intermediaries, acting as a proxy through which web service requests are routed.

To register a gateway:

1. Log in Oracle Web Services Manager to Oracle Web Services Manager the Web Services Manager Control Console.
2. Click **Add New Component**.

[Figure 10–28](#) shows a new component named BPEL Gateway:

Figure 10–28 Registering a Gateway

Add New Component:

Basic Parameters	
Component Name:	BPELGateway
Component Type:	Gateway
Container Type:	Oracle Web Services Manager
Component URL (*):	http://vmisra-pc.us.oracle.com:8888/gateway
Component Groups: <i>Modify privileges</i> su1-grp da1-grp <i>Add Groups with Modify privileges</i> ca1-grp ca2-grp <i>View privileges</i> <i>Add Groups with View privileges</i> cs1-grp cs2-grp << Add Remove >>	
Register Cancel	

Note that the Component Type is Gateway, and the Component URL matches the URL of the gateway application that resides on Oracle Enterprise Manager.

- Click **Register**. Figure 10–29 shows the confirmation page which appears:

Figure 10–29 The Gateway Component ID

Enforcement Points

Component is added successfully. Id of the new Component is "C0003001".

Ok

Make a note of the component ID, and click **OK**.

- Using a text editor, open the gateway configuration file which is located in:
 Oracle_
 Home/owsm/config/gateway/gateway-config-installer.properties
- Replace the value of gateway.component.id with the gateway ID generated in Step 3.
 Save and close the file.
- Use the wsmadmin command-line tool to start the gateway.

See Also: [Section 10.3.6, "How to Install the Server Agent"](#), Step 4 for information about wsmadmin

This procedure creates a gateway in Oracle Web Services Manager. However, there are as yet no policy steps attached to this gateway.

10.4.2 What Happens When You Register a Gateway

When you register a gateway, Oracle Web Services Manager adds it to the server system registry. It also assigns a unique component ID number to the gateway; this ID identifies the component for all actions across your deployment, and in metrics collected from the gateway.

Create a systematic naming convention for all your components. Record the "friendly" name for the component along with the following details:

- Component ID
- Product version number
- Component description
- Other component-specific details relevant to that policy enforcement point

As noted earlier, a gateway is not operational until you identify the services it will protect and define the security policies to be implemented for each service.

10.4.3 How to Locate the Service WSDL

In order to create an Oracle Web Services Manager policy for a web service, it is necessary to know the WSDL location of the service. Since service WSDLs do not reside in BPEL, this information must be retrieved from another source such as a UDDI registry or OC4J.

The procedure is illustrated by obtaining the WSDL for the Credit Validation Service service in the SOA Order Booking Application from the application server.

To locate the service WSDL:

1. Log in to the Oracle Enterprise Manager 10g Application Server Control Console.
2. Expand home.
3. Click the application whose WSDL information is required. In the example, it is the web service named SOADEMO-CREDITSERVICE-CreditService-WS.
4. On the service's Application Home page, click Web Services.
5. The list of services is displayed. In the example, the only service listed is ValidateCreditCardServiceSoapHttp. Click **Test Service**.
6. Click **Test Web Service**. This brings you to the web service home page, as shown in [Figure 10-30](#):

Figure 10–30 Web Service Home Page**CreditService endpoint**

For a formal definition, please review the [Service Description](#).

Download the JavaScript Stub (BETA) for [ValidateCreditCardServiceSoapHttp](#) and see its [documentation](#).

ValidateCreditCardServiceSoapHttp

Operation: Edit As:

☒ **Reliable Messaging** ☐ Enable

☒ **WS-Security** ☐ Enable

CreditCard

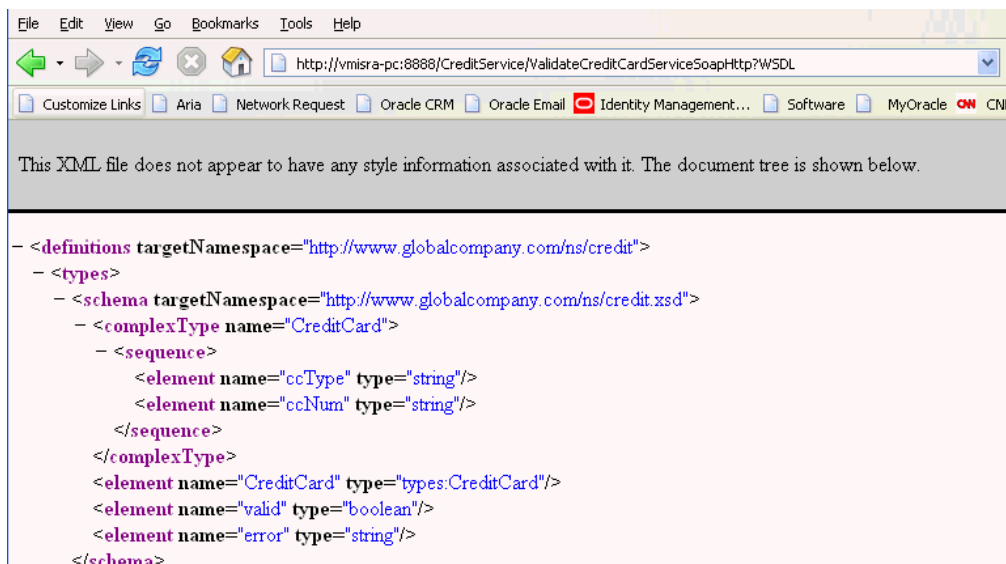
ccType		xsd:string
ccNum		xsd:string

☐ Show Transport Info

☐ Perform stress test

The home page displays the list of operations available for the web service, and links to the web service's WSDL and the service's JavaScript stub.

- Click the **Service Description** link, which opens the WSDL for the web service. [Figure 10–31](#) shows this page for the Credit Validation Service:

Figure 10–31 Service Description Page

- Your browser's address bar shows the WSDL location URL. Note the address. In the next section, this WSDL address is used to register the web service at an Oracle WSM Gateway.

Note: For an alternative approach to web service invocation, see [Section 9.7, "Testing the Web Application"](#).

10.4.4 How to Register the Web Service at the Gateway

Earlier, an Oracle Web Services Manager Gateway Oracle Web Services Manager was configured as the first step in protecting interactions between the client application and the web service.

A gateway can handle security policies for multiple web services. You must now specify the web service which the gateway Oracle Web Services Manager will secure.

To register the web service at the gateway:

1. Start at the Web Services Manager Control Console.
Click **Policy Management**, then click **Register Services**. The available gateways are displayed.
2. Locate your gateway in the list, and click the corresponding **Services** link.
3. The list of web services for the gateway are displayed. Currently there are no services defined for the example BPELGateway. Click **Add New Service**.
4. Provide this information to register the service:

Field	Value
Service Name	Gateway Service
Service Version	1.0
Service Description	BPEL Gateway Service
WSDL URL	The WSDL URL saved in Step 8 of "How to Locate the Service WSDL" on page 10-25.
Service Protocol	http(s)

Accept the default values for the other fields.

[Figure 10-32](#) shows the service details defined for the Credit Validation Service:

Figure 10–32 Adding Web Service Details to a Gateway

Step 1 of 2

Add New Service to "BPELGateway"

Service Details	
Service Name (*):	Gateway Service
Service Version (*):	1.0
Service Description:	BPEL Gateway Service
WSDL URL:	litService/ValidateCreditCardServiceSoapHttp?WSDL
Service Protocol(*):	☒ HTTP(S) ☐ JMS(SSL) ☐ IBM MQSeries ☐ Custom
Custom Protocol Step Template Id:	

Service Groups:

Modify privileges

su1-grp
da1-grp

View privileges

<< Add

Remove >>

Modify privileges

sa1-grp
sa2-grp

Add Groups with View privileges

ss1-grp
ss2-grp

5. Click **Next**. A second page appears, displaying the service protocol parameters.
6. Accept the defaults, and click **Finish**.
7. A confirmation page appears, informing you that the service has been added. Click **OK**.
8. The list of services defined at the gateway is displayed. The service needs to be committed by clicking the **Commit** link.

When you receive confirmation that policies have been committed, click **OK**.

Note: Although the commit action updated the gateway with the new service details, no security policy steps have been configured yet. Configuring an encryption policy step for the service at the gateway, and a decryption policy step at the server agent, are the subject of [Section 10.4.8, "How to Define Encryption and Decryption Policy Steps"](#).

9. From the list of services defined at the gateway, click the View Details icon of the gateway service. The service details page appears, as shown in [Figure 10–33](#):

Figure 10-33 Accessing Service Details

Details of Service: "SID0003001"

Client Access URLs	
Service URL:	http://vmisra-pc.us.oracle.com:8888/gateway/services/SID0003001
Service WSDL URL:	http://vmisra-pc.us.oracle.com:8888/gateway/services/SID0003001?wsdl
Basic Parameters	
Service Name:	Gateway Service
Service Version:	1.0
Service Description:	BPEL Gateway Service
Service WSDL:	WSDL
Service Protocol:	View Protocol Parameters

Make a note of the Service WSDL URL. This WSDL will be used in the next procedure to redirect credit rating service clients to the gateway.

10.4.5 How to Redirect Clients to the Gateway

To protect the web service with Oracle Web Services Manager an Oracle Web Services Manager Gateway, it is necessary to ensure that the service cannot be invoked directly. To achieve this, the service WSDL must be updated to point to the gateway.

For example, one can modify the WSDL of the Credit Validation Service so that it points to the gateway that was created earlier (described in "[How to Register the Gateway](#)" on page 10-23).

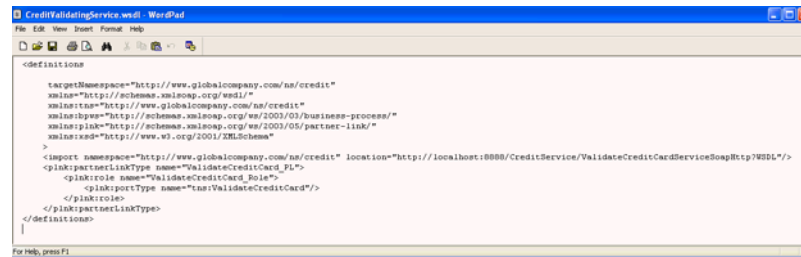
Note: Security can be extended by disabling the original URL. Publishing the new, secure URL to a Universal Description, Discovery, and Integration (UDDI) registry simplifies service discovery and makes it available for invocation by other applications.

To redirect clients to the gateway:

1. Launch JDeveloper.
2. In the Applications Navigator, locate and expand the application file (for example, SOADEMO/SOAOrderBooking).
3. Expand Integration Content.
4. Double-click the .bpel file (for example, SOAOrderBooking.bpel).
5. Scroll down the Partner Links. Expand the service of interest. Right-click the service icon and select **Edit**.
6. In the Edit Partner Link dialog, note the location of the WSDL File for the service.
7. Open the WSDL file in Oracle JDeveloper (or any text editor). To view the file in the Oracle JDeveloper WSDL Editor, in the Edit Partner Link dialog under WSDL Settings, click the **Browse WSDL Files from Local File System** icon, locate the file, and click **Open**.

Figure 10–34 shows a portion of the Credit Validation Service WSDL file:

Figure 10–34 Web Service WSDL File



8. The location attribute of the import tag contains the web service WSDL URL. Replace this with the gateway service WSDL URL.

The gateway service WSDL URL can be obtained in the Web Services Manager Control Console by clicking **Policy Management**, then **Register Services**. Click **Services** for the relevant gateway, and click **View Details** for the service.

For example, the URL for the gateway protecting the Credit Validation Service is shown in Figure 10–35:

Figure 10–35 Locating the Service WSDL URL

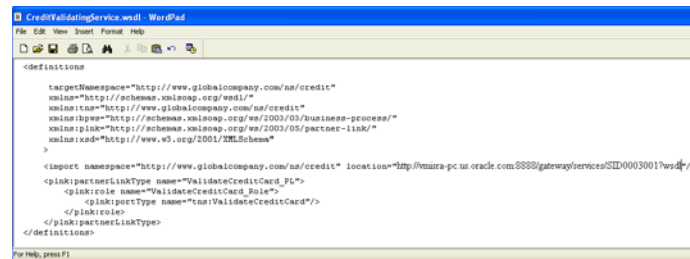
Details of Service: "SID0003001"

Client Access URLs	
Service URL:	http://vmisra-pc.us.oracle.com:8888/gateway/services/SID0003001
Service WSDL URL:	http://vmisra-pc.us.oracle.com:8888/gateway/services/SID0003001?wsdl

Basic Parameters	
Service Name:	Gateway Service
Service Version:	1.0
Service Description:	BPEL Gateway Service
Service WSDL:	WSDL
Service Protocol:	View Protocol Parameters

- Copy the service WSDL URL and paste it into the `location` field in the service WSDL file. [Figure 10–36](#) shows the file after it has been updated:

Figure 10–36 The Web Service WSDL File After Updating the Location



- Save the file. Compile and redeploy the process.

10.4.6 How to Create the Certificate Keystore

In order to perform encryption Oracle Web Services Manager and decryption, Oracle Web Services Manager keystores that contain the certificates for both the gateway and agent. In this section the `keytool` utility is invoked to generate the keystores needed.

To create a certificate keystore:

- Open a command prompt. It does not have to be the BPEL Developer Prompt; any command window will suffice.

Navigate to your `JAVA_HOME/bin` directory.

- Run the `keytool` command using the parameters needed to create a keystore. The command is one continuous line, even though it may wrap on multiple lines, as shown in [Figure 10–37](#):

Figure 10–37 The keytool Command

```

C:\Demo>keytool -keystore orderbooking.jks -genkey -keyalg RSA -alias orderbookin
g -dname "cn=order booking, ou=BPEL, o=globalcompany.com, L=New York, ST=New Yo
rk"
Enter keystore password: selecao123
Enter key password for <orderbooking>
    <RETURN if same as keystore password>: selecao123
C:\Demo>
  
```

Note: You can run the `keytool` command without any parameters to display command options.

When specifying the location of the keystore with the `-keystore` parameter value, a path can be supplied; there is no specific path requirement and the keystore can reside anywhere on the system.

The example in [Figure 10–37](#) creates a keystore named `orderbooking.jks` in the current `C:\Demo` directory.

3. You will be asked a series of questions. After you respond to the questions, the store is created.

Make a note of the information you provide on the command line and in the subsequent dialog, because you will need this information later when the gateway policy steps are defined.
4. Close the command window.

10.4.7 What You May Need to Know About Certificate Keystores

The preceding discussion of keystores illustrated setting up a single keystore on the gateway (client side). In a realistic scenario, certificate keystores are required on both the client side and the server side.

The process by which message security is implemented drives the specific configuration details. For flexibility, Oracle Web Services Manager provides a range of policy steps. For example, the message can be encrypted and signed in separate policy steps, in which case decryption and signature verification must be performed separately in reverse order.

For more information, see:

- The *Oracle Web Services Manager Administrator's Guide*. The appendix on Oracle Web Services Manager Policy Steps provides details about the usage and context of specific encryption, decryption, and signing steps.
- The *Oracle Web Services Manager Deployment Guide*. The chapter on Integrating with Oracle Business Process Execution Language describes how to install certificates to facilitate encryption and decryption, and signing and verification.

10.4.8 How to Define Encryption and Decryption Policy Steps

Policy steps must next be set up to 1) encrypt Oracle Web Services Manager the messages Oracle Web Services Manager being sent by the Oracle BPEL Process Manager client to the Oracle WSM Gateway, and 2) decrypt the messages received by the Oracle WSM Server Agent (in the SOA Order Booking Application, this agent "wraps around" the Credit Validation Service).

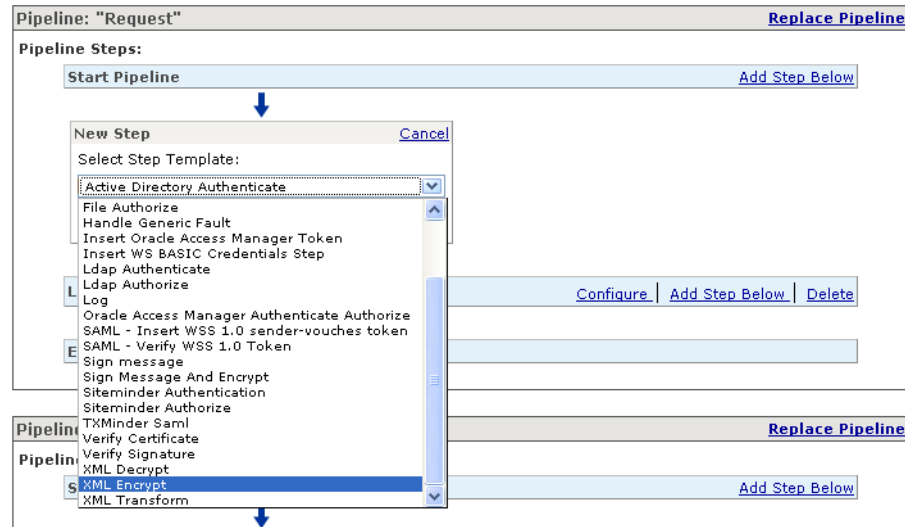
To define the encryption policy step:

1. Log in to Oracle Web Services Manager the Web Services Manager Control Console.
2. Select **Policy Management**, then click **Register Services** to list the gateways.
3. Click the **Services** link for the gateway of interest. The list of services for the gateway is displayed.
4. Click the **Edit** icon for the service of interest. The service details are displayed.

In BPELGateway, which was set up in [Section 10.4.1, "How to Register the Gateway"](#), the only configured service is the BPEL Gateway Service.
5. In Service Policy, click **Modify Policy**. The policy pipeline page appears.
6. The request and response pipelines of the policy include, by default, a Log step that enables SOAP messages to be recorded at a policy enforcement point. In the Request pipeline, choose **Add Step Below** on the Start Pipeline to add a new step below the Log step.

7. A New Step dialog box appears, as shown in [Figure 10–38](#). From the list, select XML Encrypt:

Figure 10–38 Adding an XML Encrypt Policy Step



Click **OK** in the dialog box. The encryption step appears in the Request pipeline.

8. Click the **Configure** link for the XML Encrypt step you just created. The Configure pipeline step detail page appears, as shown in [Figure 10–39](#):

Figure 10–39 Adding Details for the XML Encrypt Step

Pipeline: "Request"

Configure pipeline step

Pipeline Step Name: XML Encrypt

XML Encrypt		Environmental Properties	
Basic Properties			
Enabled (*)	boolean	Default	Value
		true	☒ true ☐ false
Encryption Properties			
Keystore location (*)	string	Type	Default
	C:/Demo/orderbooking.jks		Value
Encrypt Keystore Type (*)	string	jks	jks
Keystore password	string		*****
Decryptor's public-key alias (*)	string		orderbooking
Encryption Algorithm (*)	string	3DES	AES-128
Key Transport Algorithm (*)	string	RSA-1_5	RSA-1_5
Encrypted Content (*)	string	BODY	BODY
Encrypt XPATH Expression	string		
Encrypt XML Namespace	string[]		

Ok Cancel

Information that you need to provide on this page includes:

- Keystore location (this is the location that was specified when the keystore was created)
- Encryption keystore type (jks in the example)
- Keystore password
- Decryptor's public key alias
- Encrypted content (BODY)
- Additional details about the encryption

9. Click **OK**.

This completes the steps to encrypt the content that is being sent to the web service from the Oracle BPEL Process Manager client.

To define the decryption policy step:

1. In the Web Services Manager Control Console, display the component list by clicking **Policy Management**, then clicking **Manage Policies**.
2. From the list of Oracle Web Services Manager components, find the server agent associated with the web service. For example, the Authentication Agent is the server agent protecting the Credit Validation Service.

Click **Policies** and click the **Edit** icon corresponding to the default policy.

3. The agent policy appears. Scroll down to the Request pipeline, and click **Add Step Below** on the Start Pipeline to add a new step above the Log step.
4. From the New Step list, select XML Decrypt and press **OK** below the dialog box.
5. Click the **Configure** link for the new XML Decrypt step you just created. The Configure pipeline step detail page appears, as shown in [Figure 10–40](#):

Figure 10–40 Adding Details for the XML Decrypt Step

Pipeline: "Request"

Configure pipeline step

Pipeline Step Name: XML Decrypt

XML Decrypt		Environmental Properties	
Basic Properties	Type	Default	Value
Enabled (*)	boolean	true	☒ true ☐ false
XML Decryption Properties			
Keystore location (*)	string		C:/Demo/orderbooking.jks
Decrypt Keystore Type (*)	string	jks	jks
Keystore password	string		*****
Decryptor's private-key alias (*)	string		orderbooking
Decryptor's private-key password	string		*****
Enforce Encryption (*)	boolean	true	☒ true ☐ false

Ok Cancel

Information that you need to provide on this page includes:

- Keystore location
 - Decryption keystore type (jks)
 - Keystore password
 - Decryptor's private key alias
 - Decryptor's private key password
 - Enforce encryption (true)
6. Click **OK**, then click **Next**, and click **Save**.
 7. On the Policy Set page, click **Commit** at the top of the page.

This completes the steps to decrypt the message being received on the web service end.

10.4.9 How to Test Message Encryption

Encryption and decryption of the content, as messages flow between the Oracle BPEL Process Manager process and the web service, can be observed by opening a TCP tunnel and watching the message traffic.

Note: You can also use the Oracle JDeveloper HTTP Analyzer to see the contents of SOAP messages passed when a proxy connects to a web service and the service returns a response. See [Section 5.4, "Debugging, Testing and Analyzing Web Services in JDeveloper"](#).

To observe the message traffic:

1. Open the BPEL Developer Prompt from the Windows Start menu. For example:
From the **Start** menu, click **All Programs**, then click **Oracle BPEL Process Manager 10.1.2**, and click **Developer Prompt**.
2. At the developer prompt, enter the command:

```
obtunnel
```


This starts the TCP tunnel.
3. In the TCPMonitor Console, check that the Port and Host fields have the correct values, for example 1234 and 8888. If they do not, click **Stop** to set the values, and click **Start**.
4. Run the application to initiate message flow. For example, run SOA Order Booking Application to submit a customer order. The flow can be observed in the Oracle BPEL Process Manager Console Flow Viewer under the In-Flight Process Instances.
5. Look at the TCP tunnel window again. The top window shows that the message body is encrypted. This is how the message went across the network. The Oracle Web Services Manager Server Agent on the other end will decrypt it for the web service.

Deploying SOA Applications with Enterprise Manager

This chapter describes how to deploy J2EE-compliant applications and standalone modules into OC4J using Oracle Enterprise Manager Application Server Control.

This chapter includes the following sections:

- [Section 11.1, "Introduction to Deploying SOA Applications with Oracle Enterprise Manager Application Server Control"](#)
- [Section 11.2, "Preparing for Deployment"](#)
- [Section 11.3, "Deploying the Application with Oracle Enterprise Manager Application Server Control"](#)
- [Section 11.4, "Performing Post-Deployment Tasks"](#)

11.1 Introduction to Deploying SOA Applications with Oracle Enterprise Manager Application Server Control

The Oracle Enterprise Manager Application Server Control console provides a web-based user interface for completing deployment-related tasks including:

- Deploying, undeploying, and redeploying standalone modules (WARs, EARs, etc.) into the selected OC4J instance
- Creating and editing reusable deployment plans
- Setting application-specific security and application clustering configurations
- Importing shared libraries for use by the application, as well as modifying the list of inherited libraries
- Deploying to a specific OC4J instance or to a group of instances within an Oracle Application Server cluster

Because it provides such robust functionality, the Oracle Enterprise Manager Application Server Control console should be your first choice for performing deployment-related tasks.

11.2 Preparing for Deployment

Before you deploy an SOA application, you must plan the deployment process and complete the following tasks:

- Define the requirements for deploying the SOA application
- Configure and set up load balancing within a cluster
- Create data sources (Optional)
- Configure and deploy standalone resource adapters (Optional)

Note: You can create data sources and configure standalone resource adapters after the application has been deployed.

11.2.1 How to Define Deployment Requirements

Before you start using the Application Server Control to deploy your SOA application, ensure that:

- You have already created an application archive (For example, a `.ear` or `.war` file). For more information on creating an application archive, refer to the *Oracle Containers for J2EE Configuration and Administration Guide*.
- You have installed Oracle Application Server 10g Release 3 (10.1.3x) on one or more hosts.
- You have previously logged into the Oracle Enterprise Manager Application Server Control console. For more information, see [Section 11.3, "Deploying the Application with Oracle Enterprise Manager Application Server Control"](#).

11.2.2 How to Configure Load Balancing Within a Cluster

The term *load balancing* refers to the process of distributing incoming service requests over server instances within a cluster. Load balancing in an Oracle Application Server cluster is managed by the `mod_oc4j` module of Oracle HTTP Server (OHS). In this configuration, the OHS instance acts as a front-end listener for incoming HTTP/HTTPS requests; `mod_oc4j` then routes each request to an OC4J instance serving the requested application. For more information on load balancing in an Oracle Application Server cluster, refer to *Oracle Containers for J2EE Configuration and Administration Guide*.

11.2.2.1 OC4J Groups

A group is a loosely synchronized set of like-named OC4J instances that belong to the same cluster topology. Specific configuration options can be executed simultaneously on all OC4J instances in the group.

You can create a group by creating two OC4J instances that have the same name and reside in the same cluster. For example, suppose you install two Oracle Application Server instances and configure them as a cluster. If you create an OC4J instance called `home` in one instance, and then you create an instance called `home` in the second instance, a new group called `home` automatically appears in the **Groups** section of the Cluster Topology page.

After you create a group, you can perform these common administrative tasks automatically across multiple OC4J instances:

- Process management operations, such as start, stop, or restart
- Deployment operations, such as deploy, undeploy, and redeploy
- JDBC management operations, such as creating, modifying, or removing JDBC data sources and connection pools
- JMS Provider operations, such as creating, modifying, or removing JDBC data sources and connection pools

For more information on managing groups, refer to *Oracle Application Server Administrator's Guide*.

11.2.2.2 Multiple JVMs

OC4J executes on the Java Virtual Machine (JVM) of the standard Java Development Kit (JDK). By default, each OC4J instance uses one JVM. However, you can configure an OC4J instance so it runs on multiple JVMs.

When you configure an OC4J instance to run on multiple JVMs, the OC4J instance is essentially running on multiple processes. This can improve performance and provide a level of fault tolerance for your deployed applications. However, multiple JVMs also require additional hardware resources to run efficiently.

Note: Application Server Control (represented by the `ascontrol` application) cannot run on an OC4J instance that is running multiple JVMs. Ensure that you do not configure multiple JVMs on the OC4J instance that is hosting the active `ascontrol`.

When you run multiple JVMs, consider configuring your deployed application so they take advantage of application clustering. Application clustering ensures that state information is replicated to the different instances of your application running in each JVM.

It is also useful to compare the use of multiple JVMs with the use of application server clusters and groups. For example, when you configure an OC4J instance to run on multiple JVMs, the multiple processes are running on the same host. If the host goes down, all the JVM processes will go down. For more information, see *Oracle Containers for J2EE Configuration and Administration Guide*.

11.2.2.3 Application Clustering (State Replication)

In addition to clusters and groups, Oracle Application Server 10g Release 3 (10.1.3) introduces the concept of application clustering, which provides state replication and load balancing for applications within your cluster topology.

Within a cluster, you can configure clustering for selected applications that are deployed across the cluster. Application clustering offers the following features:

- You can configure clustering for specific applications or globally by configuring clustering for the `default` application in an OC4J instance. Other applications deployed to the instance automatically inherit the clustering characteristics of the `default` application.
- You can configure clustering for an application at deployment time, or later, after you deploy the application.

- You can select from the following replication methods:
 - Peer-to-peer replication
 - Multicast replication
 - Database replication

For more information on application clustering, refer to the *Oracle Containers for J2EE Configuration and Administration Guide*.

11.2.3 How to Create Data Sources

A data source is a vendor-independent encapsulation of database server connection. A data source instantiates an object that implements the `javax.sql.DataSource` interface.

You can define any type of data source using Oracle Enterprise Manager Application Server Control Console. You can configure and use these data sources in your Oracle Application Server Containers for J2EE (OC4J) application.

Note: You can create the data source after your application has been deployed.

To create data sources and connection pools for a selected OC4J instance:

1. Navigate to the OC4J Home page for the OC4J instance.
2. Click **Administration** to display the OC4J Administration page, which contains a table listing the various administration tasks you can perform for this OC4J instance.
3. If necessary, expand the **Services** section of the table.
4. Click the task icon in the **JDBC Resources** row of the table to display the JDBC Resources as shown in [Figure 11–1](#).

Figure 11–1 JDC Resources

The screenshot shows the Oracle Enterprise Manager 10g Application Server Control interface. The breadcrumb trail indicates the path: Cluster Topology > Application Server: 060725basic.stact52.us.oracle.com > OC4J: home > JDBC Resources. The application selected is SOADEMO. The page is titled "JDBC Resources" and shows two main sections: "Data Sources" and "Connection Pools".

Data Sources Table:

Name /	Application	JNDI Location	Connection Pool	Managed by OC4J	Test Connection	Delete
"dev-connection-managed-soademo"	SOADEMO	jdbc/soademoDS	"dev-connection-pool-soademo"	✓	[Test]	[Delete]
"dev-connection-native-soademo"	SOADEMO	jdbc/soademoCoreDS			[Test]	[Delete]

Connection Pools Table:

Name /	Application	Connection Factory Class	Monitor Performance	Test Connection	Refresh Connection Pool	Delete
"dev-connection-pool-soademo"	SOADEMO	oracle.jdbc.pool.OracleDataSource	[Monitor]	[Test]	[Refresh]	[Delete]

5. Click **Help** for more information about viewing, creating, and deleting data sources and connection pools.

11.2.4 How to Deploy and Configure Standalone Resource Adapters

You can use the Oracle Enterprise Manager Application Server Control console to deploy, monitor, and configure standalone resource adapters, which are packaged and deployed as .rar files. In addition, you can also manage embedded resource adapters, which are deployed as part of an application.

Note: You can deploy and configure standalone resource adapters after the application has been deployed.

Before an application component can use resource adapter connections to an EIS, you must create one or more connection factories. Creating and configuring a connection factory involves the following tasks:

- Identify the connection interface.
- Bind the connection to JNDI.
- Define the connection pooling characteristics.
- Provide any configuration properties for the specific connection factory.

To use the Application Server Control Console to create a new connection factory:

1. Navigate to the Resource Adapter Home page.
2. Click **Connection Factories** to display the Connection Factories page as shown in [Figure 11–2](#). This page lists the connection factories and shared connection pools defined for the selected resource adapter.

Figure 11–2 Connection Factories

ORACLE Enterprise Manager 10g
Application Server Control

Cluster Topology > Application Server: 060725basic.stac252.us.oracle.com > OC4J: home > Application: default >

Resource Adapter: Database Adapter

Page Refreshed Jul 28, 2006 11:06:26 AM PDT

Home Connection Factories Administered Objects Administration

Connection factories exposed by the resource adapter are used by application components to obtain connections to the EIS. This table lists the connection factories currently configured for this resource adapter.

(Create)

JNDI Location	Connection Factory Interface	Connection Pool Used	Monitor Connection Factory	Delete
eis/DB/BPELSamples	javax.resource.cci.ConnectionFactory	None		
eis/DB/DBConnection1	javax.resource.cci.ConnectionFactory	None		
eis/DB/OLite	javax.resource.cci.ConnectionFactory	None		
eis/DB/soademo	javax.resource.cci.ConnectionFactory	None		

Shared Connection Pools

Shared connection pools are connection pools configured for this resource adapter, that can be shared amongst connection factories that use the same connection factory interface. A shared connection pool that's currently being used by a connection factory cannot be deleted.

(Create)

Name	In Use	Monitor Connection Pool	Cleanup Connections	Delete
No shared connection pools found				

Home Connection Factories Administered Objects Administration

3. Click **Create** at the top of the connection factory table and follow the instructions in the Create Connection Factory wizard.

Standalone resource adapters can be deployed just like applications and application modules. Multiple applications can then use a single standalone resource adapter to connect to an enterprise information system (EIS).

To deploy a standalone resource adapter:

1. Navigate to the OC4J Home page for the instance.
2. Click **Applications** to display the OC4J Applications page.
3. Choose Standalone Resource Adapters from the View drop-down menu to display the Standalone Resource Adapters as shown in Figure 11–3. This page lists the standalone resource adapters that have been deployed to the OC4J instance.

Figure 11–3 Standalone Resource Adapters

ORACLE Enterprise Manager 10g
Application Server Control

Cluster Topology > Application Server: 060725basic.stac252.us.oracle.com >
OC4J: home

Page Refreshed Jul 28, 2006 11:04:15 AM PDT

Home Applications Web Services Performance Administration

This page shows the J2EE applications and application components (EJB Modules, WAR Modules, Resource Adapter Modules) deployed to this OC4J instance.

View Standalone Resource Adapters

Start Stop Undeploy Redeploy Deploy

Select Resource Adapter	Resource Adapter Module	Status	EIS Type
☐ Aq Adapter	AqAdapter	↑	Oracle Advanced Queueing
☒ Database Adapter	DbAdapter	↑	Database
☐ File Adapter	FileAdapter	↑	Local File System
☐ FTP Adapter	FtpAdapter	↑	FTP
☐ JMS Adapter	JmsAdapter	↑	Java Message Server
☐ Native MQSeries Adapter	MQSeriesAdapter	↑	MQSeries
☐ Oracle Applications Adapter	AppsAdapter	↑	Oracle Applications
☐ OracleASjms	OracleASjms	↑	OracleAS JMS

Home Applications Web Services Performance Administration

4. Click **Deploy**.

Follow the instructions in the Deployment wizard. The wizard prompts you for the location of the resource adapter `.rar` file and prompts you deployment options.

11.3 Deploying the Application with Oracle Enterprise Manager Application Server Control

You can deploy, undeploy, or redeploy an application or a module using Oracle Enterprise Manager Application Server Control console.

The Application Server Control Console is installed and configured automatically when you install OC4J using the Oracle Universal Installer. The console is started with all other installed Oracle Application Server components, using the OPMN command-line tool, `opmnctl`, which is installed in the `ORACLE_HOME/opmn/bin` directory on each server node. Start all the installed components by issuing the following command:

```
http://ohs_host_address:port/em
```

where:

- `ohs_host_address` is the address of the OHS host machine.
- `port` is an HTTP listener port assigned to OHS by OPMN. Run the following `opmnctl` command on the OHS host machine to get the list of assigned listener ports from OPMN.

```
opmnctl status -l
```

Supply the port designated as `http1` in the OPMN status output as the value for `port`.

```
HTTP_Server | HTTP_Server | 6412 | Alive | 1970872013 | 1
6396 | 0:48:01 | https:4443,http2:722,http1:7779
```

11.3.1 How to Deploy the Application

All applications in the SOA infrastructure that use an archive file (`.ear` or `.war`) can be deployed using Enterprise Manager Application Server Control console. This console has a three-page deployment wizard that provides a streamlined, user-friendly deployment process.

You can deploy a new application, module, or standalone resource adapter. When you install or create a new OC4J instance, two applications are automatically deployed to the OC4J instance:

- The default application (`default`): This is the parent application for all applications that you deploy on the OC4J instance. All applications deployed on an OC4J instance inherit the attributes and properties of the `default` application.
- The Application Server Control application (`ascontrol`): The `ascontrol` application represents the Application Server Control, which is automatically deployed on all OC4J instances like any other J2EE application.

Note: If the HTTP session times out due to browser inactivity while you are using the deployment wizard, you will have to restart the deployment process.

To deploy the application:

1. Navigate to the **Cluster Topology** page as shown in [Figure 11–4](#).

Figure 11–4 Cluster Topology

ORACLE Enterprise Manager 10g
Application Server Control

Setup Logs Help Logout

Cluster Topology

Page Refreshed Jul 26, 2006 4:15:39 PM PDT • View Data Manual Refresh

Overview

Hosts 1 Application Servers 1
OC4J Instances 1 HTTP Server Instances 0

Members

View By Applications

[Expand All](#) | [Collapse All](#)

Name	Status	OC4J Instance	Application Server	Host	Context Roots	Routing Enabled	Clustering Enabled	JGroup	Replication Protocol
▼ Applications									
▶ ascontrol									
▶ ccore									
▶ coreman									
▶ CustomerService									
▶ datatags									
▶ default									
▶ esb-dt									
▶ esb-rt									
▶ gateway									
▶ hw_services									
▶ javasso									
▶ orabpel									
▶ orainfra									
▶ policymanager									
▶ ruleauthor									
▶ rulehelp									
▼ SOADEMO									
▶ SOADEMO	↑	home	060725basic.stacz52.us.oracle.com	stacz52/SOADEMO.rmiTunnel		✓			
▶ SOADEMO-CREDITSERVICE- CreditService-WS									
▶ SOADEMO-RAPIDSERVICE- RapidService-WS									
▶ WSIL-App									

↑ Indicates the active ASControl instance.

⚡ TIP Only applications on running OC4Js are shown.

Groups

A group is a collection of OC4J instances. Certain common management tasks can be performed simultaneously on all OC4J instances in a group. For more information, see [About Groups](#).

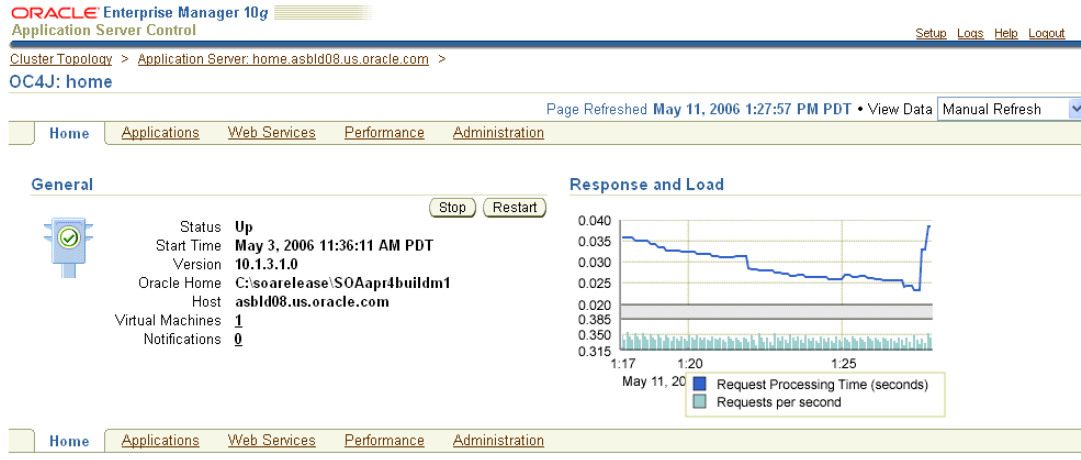
(Start) (Stop) (Delete) | (Create)

Select Name	OC4J Instance	Status	Application Server
default_group	home	↑	060725basic.stacz52.us.oracle.com

Administration

- Cluster MBean Browser
- Routing ID Configuration
- Java SSO Configuration
- Topology Network Configuration
- Runtime Ports

2. Click the link for the target instance you want to deploy the application to.
The Home page for the OC4J instance is displayed as shown in [Figure 11–5](#).

Figure 11–5 OC4J Home Page

3. Click **Applications**.

4. Click **Deploy**.

The Deploy: Select Archive page is displayed as shown in Figure 11–6.

Figure 11–6 Deploy: Select Archive

5. Select the archive to upload to the OC4J server in the first page of the wizard. The archive file that you select can be a J2EE application (EAR file), web module (WAR file), EJB module (EJB JAR file) or resource adapter module (RAR file). Enter the local path to the archive file as follows:

/applications/soademo/soademo.ear (On Unix)

\\soademo\\soademo.ear (On Windows)

You can also click **Browse** and select the location of the file on your local disk or on the server where the OC4J instance is running.

6. Optionally, you can specify a deployment plan.

The deployment plan is an XML file that consolidates all the OC4J-specific configuration data present in multiple OC4J-specific deployment descriptor files. This deployment plan can be applied to the archive or used as a template for a new deployment plan.

7. Select the default option to automatically create a deployment plan and click **Next**. You will see the second page of the deployment wizard, the Deploy: Application Attributes page as shown in [Figure 11-7](#).

Figure 11-7 Deploy: Application Attributes

8. The type of archive file being deployed and the location is displayed. If you selected an existing deployment plan, the name and location of the deployment plan is displayed. This field is empty if you selected the default option, which is to create a new deployment plan based on the deployment descriptors in archive file.
9. Enter a name for the application you are about to deploy. This name can contain spaces and will identify this application in the list of applications on the OC4J Applications page.
10. Select the application that will be used as the parent application for the application you are about to deploy.

For example, you can select the `default` application, which is available in every OC4J instance. Child applications see the namespace of their parent applications. This setting is used to share services, such as EJBs, among multiple applications.

11. If deploying a web application, bind the application to the web site that will be used to access it. Select a web application from the drop-down menu.

A `<web-app>` element is added to the specified web site XML file (for example, `http-web-site.xml`), indicating the application name, the web application name, and the context path. For example:

```
<web-app application="utility"
  name="utility_web"
  root="/utilroot"
  load-on-startup="false"
  max-inactivity-time="no shutdown"
  shared="false"/>
```

12. Finally, you can view or modify the context root for each of the web modules packaged within the application you are deploying.

The Web module context root is appended to the URL used to access the application through a Web browser. For example, if you enter the context root as /soademo, the module can be accessed with the following URL:

`http://node1.company.com:777/soademo`

13. Click **Next** to go to the next step in the Deployment Wizard.
14. You will see the Deploy: Deploy Settings page as shown in Figure 11–8.

Figure 11–8 Deploy: Deployment Settings

ORACLE Enterprise Manager 10g
Application Server Control

Help Logout

Select Archive Application Attributes **Deployment Settings**

Deploy: Deployment Settings

Cancel Back Step 3 of 3 Deploy

Archive Type **J2EE Application (EAR file)**
Archive Location **C:\documentation\SOADEMO.ear**
Deployment Plan **Creating a new plan**

Application Name **soademo**
Parent Application **default**
Bind Web Module to Site **default-web-site**
Context Root **soademo**

Deployment Tasks

The table below provides a set of common deployment tasks you might want to perform for this application. Only those tasks that apply to the current application are enabled.

Task Name	Go To Task	Description
Map Environment References		Map any environment references in your application (for example, data sources) to physical entities currently present on the operational environment.
Select Security Provider		A security provider acts as the source for available users and groups when mapping security roles.
Map Security Roles		Map any security roles exposed by your application to existing users and groups. The list of users and groups is obtained from the security provider you selected for this application.
Configure EJBs		Configure the Enterprise JavaBeans in your application.
Configure Clustering		Configure clustering of your application.
Configure Class Loading		Manipulate the classpath of your application.

Advanced Deployment Plan Editing

Click Edit Deployment Plan to set more advanced deployment options. Edit Deployment Plan

Save Deployment Plan

After you make changes, you can save the deployment plan to your local disk. You can then use the saved deployment plan to redeploy this application later. Save Deployment Plan

Cancel Back Step 3 of 3 Deploy

15. Click **Deploy** to deploy the application.

Note: When you click **Deploy**, the deployment plan is sent to the OC4J server with the archive. The deployment process will continue even if the Web browser is closed.

11.3.2 What Happens When You Deploy the Application

When you deploy an application packaged within an .ear file, the following occurs:

1. If the application is being redeployed, the existing installation is first undeployed from OC4J.
2. OC4J copies the EAR file to the master deployment directory, which defaults to the `ORACLE_HOME/j2ee/home/applications/` directory.

3. OC4J opens and parses the `application.xml` file packaged within the `.ear` file. This file is a standard J2EE descriptor that lists all of the modules contained within the `.ear` file. OC4J notes these modules and initializes the EAR environment.
4. OC4J reads the module deployment descriptors for each module type, Web module (WAR), EJB module, connector module, or client module into memory. The JAR and WAR file environments are also initialized.
5. OC4J reacts to the configuration details contained in both the J2EE deployment descriptors and any OC4J-specific deployment descriptors. OC4J notes any J2EE component configurations that require action by OC4J, such as wrapping EJBs with their interfaces.
6. OC4J writes out new OC4J-specific configuration files to the `ORACLE_HOME/j2ee/home/application-deployments/app_name` directory, according to the contents of the deployment plan. Note that if one or more OC4J-specific deployment descriptors were supplied, you may notice that OC4J added additional elements to the generated files.

Any generated classes, such as EJB interface wrapper classes, are compiled and put into new subdirectories under this directory. For example, EJB wrapper classes are generated within an archive named `deployment-cache.jar` within the `ORACLE_HOME/j2ee/home/application-deployments/app_name/jar_name.jar` directory, where `jar_name.jar` corresponds to the name of a deployed EJB JAR.

7. Finally, OC4J updates the `OC4J server.xml` configuration file with the notation that this application has been deployed.

11.3.3 How to Deploy to OC4J Instances Within a Cluster

The Application Server Control Console enables you to deploy an application to a specific OC4J instance or to a "group" of OC4J instances within an Oracle Application Server Cluster. A group is a synchronized set of OC4J instances that belong to the same cluster topology. Configuration operations can be executed simultaneously on all OC4J instances in the group.

To deploy the application to a group of OC4J instances within a cluster:

1. Click **Cluster Topology** on the Application Server Control Console Home page. You will see the Cluster Topology page.

This page displays the following: You can see that there is a group named `soa_group` listed under the Groups section in the Cluster Topology page.

 - All Oracle Application Server instances that are currently part of the cluster.
 - The active OC4J instances within each Oracle Application Server instance.
 - The applications deployed into each OC4J instance.
2. To deploy to a group of OC4 instances, click the common instance name shared by the group under Groups at the bottom of the page.
3. Deploy the application by following the instructions as listed in [Section 11.3.1, "How to Deploy the Application"](#).

11.4 Performing Post-Deployment Tasks

This section describes some of the typical post-deployment tasks that need to be performed after the SOA application has been deployed. Post-deployment tasks include:

- Setting up Notifications
- Redeploying the SOA Application
- Undeploying the SOA Application

11.4.1 How to Set Up Notifications

Many of the MBeans packaged with OC4J include the ability to generate notifications triggered by a state change registered by the MBean. For example, you can subscribe to a notification that will alert you when a selected application has stopped.

To subscribe to specific JMX event notifications:

1. Navigate to the OC4J Home page.
2. Click **Administration**.
Enterprise Manager displays the OC4J Administration page, which lists all the administration tasks you can perform for this instance.
3. Click the Task icon in the **Notification Subscriptions** row of the table.
Enterprise Manager displays the Notification Subscriptions page, which lists all the available notifications, grouped by application and service.
4. Select the check box in the **Subscribe** column for every notification you want to receive.

11.4.2 How to Redeploy the Application

After you deploy the application, you may need to undeploy the application or redeploy a newer version of the application. Application Server Control accommodates these scenarios by offering the ability to redeploy an application.

To redeploy an application:

1. Navigate to the OC4J Home page for the OC4J instance.
2. Click **Applications**.
3. Select the application and click **Redeploy**.

11.4.2.1 What Happens When You Redeploy the Application

Redeploying a J2EE application packaged within an EAR file prompts OC4J to undeploy the previous instance of the J2EE application, including any embedded resource adapters packaged with the application.

If changes are made to any of the following configuration files as part of redeployment, OC4J must be restarted for the changes to take effect. Otherwise, a restart of OC4J is not required after redeploying an application. This requirement applies only to configuration files that are already installed on the server.

- `data-sources.xml`
- `jms.xml`
- `rmi.xml`

The application is completely inaccessible during redeployment, and incoming requests will not be processed until the updated application is restarted by OC4J when deployment is complete.

11.4.2.2 Impact of Redeploying a Parent Application

After redeploying an application that is the parent of one or more child applications, you should ideally restart each child application. Restarting will ensure that the child applications are able to access any inherited classes or shared libraries provided through the parent.

11.4.3 How to Undeploy the Application

After you deploy the application, you may need to undeploy the application or redeploy a newer version of the application. Oracle Enterprise Manager Application Server Control accommodates these scenarios by offering the ability to undeploy an application.

To undeploy an application:

1. Navigate to the OC4J Home page for the OC4J instance.
2. Click **Applications**.
3. Select the application and click **Undeploy**.

11.4.3.1 What Happens When You Undeploy the Application

Removing a J2EE application from an OC4J instance results in the following:

- The application is removed from the OC4J runtime.
- All bindings for the Web applications are removed from all the Web sites to which the Web modules are bound.
- All application files are removed from both the `applications/` and `application-deployments/` directories.

11.4.3.2 Impact of Undeploying a Parent Applications

When an application that is the parent of one or more child applications is undeployed, the child applications are also undeployed.

This means that all of the related applications - the parent as well as its dependent applications - must be redeployed. Bear this in mind before undeploying a parent application.

11.4.4 Troubleshooting Deployment Errors

This section discusses common errors that occur during deployment

■ Interruptions During Application Deployment

If the deployment process is interrupted for any reason, you may need to clean up the `temp` directory, which by default is `/var/tmp`, on your system.

The Application Server Control Console Deployment wizard uses approximately 20 MB in swap space of the `temp` directory for storing information during the deployment process. At completion, the deployment wizard cleans up the additional files in the `temp` directory.

However, if the wizard is interrupted, it may not have the time or opportunity to clean up the `temp` directory. You must manually clean up any additional deployment files in the directory. Otherwise, the `temp` directory may run out of space and further deployment will not be possible.

You can change the `temp` directory at OC4J startup by setting the `java.io.tmpdir` command-line option to a new location. For more information on setting the system properties, refer to *Oracle Containers for J2EE Configuration and Administration Guide*.

Part III

Monitoring SOA Systems

Part III contains the following chapters:

- [Chapter 12, "Monitoring Oracle Enterprise Service Bus"](#)
- [Chapter 13, "Monitoring Oracle BPEL Process Manager"](#)
- [Chapter 14, "Monitoring Security"](#)
- [Chapter 15, "Monitoring SOA Applications with Enterprise Manager"](#)

Monitoring Oracle Enterprise Service Bus

This chapter describes how to monitor message instances across Oracle Enterprise Service Bus (ESB) using Oracle ESB Control. Error handling is also discussed in this chapter.

This chapter includes the following sections:

- [Section 12.1, "Introduction to Monitoring with Oracle ESB Control"](#)
- [Section 12.2, "Components of Oracle ESB Control"](#)
- [Section 12.3, "Using the Services View to Review and Update ESB Systems"](#)
- [Section 12.4, "Using the Services View to Review and Update ESB Services"](#)
- [Section 12.5, "Using the Maps View to Define Domain-Value Maps"](#)
- [Section 12.6, "Using the Instances View to Monitor Message Instances"](#)
- [Section 12.7, "Searching for Message Instances"](#)
- [Section 12.8, "Managing Message Instance Tracking"](#)
- [Section 12.9, "Defining Trackable Fields"](#)
- [Section 12.10, "Viewing Log Files"](#)
- [Section 12.11, "Error Handling in Message Instance Processing"](#)

12.1 Introduction to Monitoring with Oracle ESB Control

Oracle ESB Control is used at runtime to monitor message instance processing across an ESB flow in a schematic diagram, allowing you to determine the success of the processing and correct problems if they occur.

With Oracle ESB Control, you have a graphical representation of the ESB flow. You can view the connections among the various Oracle Enterprise Service Bus services in the schematic diagram. In addition, you can view system, group, and service definition details, including URLs required to call ESB services from applications that are external to ESB. You can also update and delete ESB systems, routing services, and SOAP invocation services.

Oracle ESB Control enables you to view, modify, and define routing rules for routing services, including filter expressions, transformations, and execution type (synchronous or asynchronous) to fine tune the runtime processing.

You can view error conditions within a schematic, including the messages that did not reach their destination. Error handling in Oracle ESB Control provides for resubmitting messages that can be retried.

Read this chapter to understand:

- How to monitor a message instance flow with Oracle ESB Control
- How to review and update ESB system definitions with Oracle ESB Control
- How to review and update ESB service definitions with Oracle ESB Control
- How to handle runtime errors with Oracle ESB Control

For more information about Oracle ESB Control, see "Monitoring the Enterprise Service Bus" in *Oracle Enterprise Service Bus Developer's Guide*.

12.2 Components of Oracle ESB Control

Oracle ESB Control is vertically divided into two main parts: the Service Navigation tree and the Configuration panel.

Oracle ESB Control provides three views, which you select by clicking the icons that are across the top of Oracle ESB Control:

- Services view

The Services view of Oracle ESB Control enables you to view system and service definitions, update routing rules, update endpoint properties, define trackable fields, and view a schematic diagram of the services.

- Instances view

The Instances view of Oracle ESB Control enables you to view details about message processing across an ESB system.

- Maps view

The Maps view of Oracle ESB Control enables you to create, update, and delete domain-value maps, as well as view, export, and import existing domain-value maps.

12.2.1 How to Open the Oracle ESB Control

After an ESB project has been completed and registered with the ESB Server or an integration server, you can view the project services in Oracle ESB Control.

To open Oracle ESB Control:

To open Oracle ESB Control, you need to navigate to the URL of the following form:

`http://host_name:port_number/esb/`

where *host_name* is the host computer where the ESB Server has been installed and *port_number* is the port number of the HTTP proxy server.

If the ESB installation is on Windows, you can open Oracle ESB Control by selecting **All Programs > Oracle – *Oracle_Home* > Oracle ESB 10.1.3 > ESB Control** from the desktop **Start** button, where *Oracle_Home* is the name of the Oracle home where you installed Oracle Enterprise Service Bus.

Note: Before opening Oracle ESB Control, you must have the ESB Server running. The process of starting and stopping the ESB server depends on the operating system where the ESB server is located. See *Oracle Enterprise Service Bus Installation Guide* and the Oracle Application Server installation guide for your operating system.

12.3 Using the Services View to Review and Update ESB Systems

The Services view of Oracle ESB Control enables you to view and update system definitions.

Figure 12–1 provides an example of the system Definition tab of the Services view in Oracle ESB Control.

Figure 12–1 Oracle ESB Control Services View - System Definition Tab

The screenshot shows the Oracle ESB Control interface. On the left, a 'Services' navigation pane lists various systems: DefaultSystem, BPESystem, CustomerData, CustomerQuery, CustomerHeaderSupl, CustDataSys, and Fulfillment (selected). Below Fulfillment, there are sub-items: Shipment, OrderFulfillment, FedExShipment, USPSShipment, and FulfillmentBatch. The main panel displays the 'Definition' tab for the 'Fulfillment' system. It includes a 'Reset' button and a 'Click 'Apply' to save your changes' instruction. The configuration fields are as follows:

Field	Value
* Cluster Name	esb
* Virtual Host	Application Server Cluster associated with the ESB system
* Port	localhost
	8888

Asynchronous Topic Details
 Configure the JNDI location of the topic and topic connection factory. Configure the number of listeners

Field	Value
Topic Location	OracleASjms/ESBDeferredTopic
Connection Factory Location	
Number of Listeners	1

Notification Details
 The following channels will be notified if errors occur in the ESB. Use semicolons to separate multiple entries.

Field	Value
Email ID	
Pager Number	
Mobile Number	
Phone Number	

The system Definition tab of the Services view displays the name, cluster name, virtual host, port, asynchronous topic details, and notification details for the ESB system.

You can update the system definitions. For example, you can enter contact information in the notification channels under notification details. With the notification channels you can specify the people to be notified in the event that an error occurs in the processing of message instances.

12.3.1 How to View and Modify System Definitions

You can view and update an ESB system definition in the Services view of Oracle ESB Control. When you select a system in the navigation tree of the services panel, the system Definition tab displays.

To view and modify system definitions:

1. Open the Services view of Oracle ESB Control.
2. In the **Services** navigation tree, select the ESB system for which you want to set notification channels.

The Definition tab appears similar to [Figure 12-1](#).

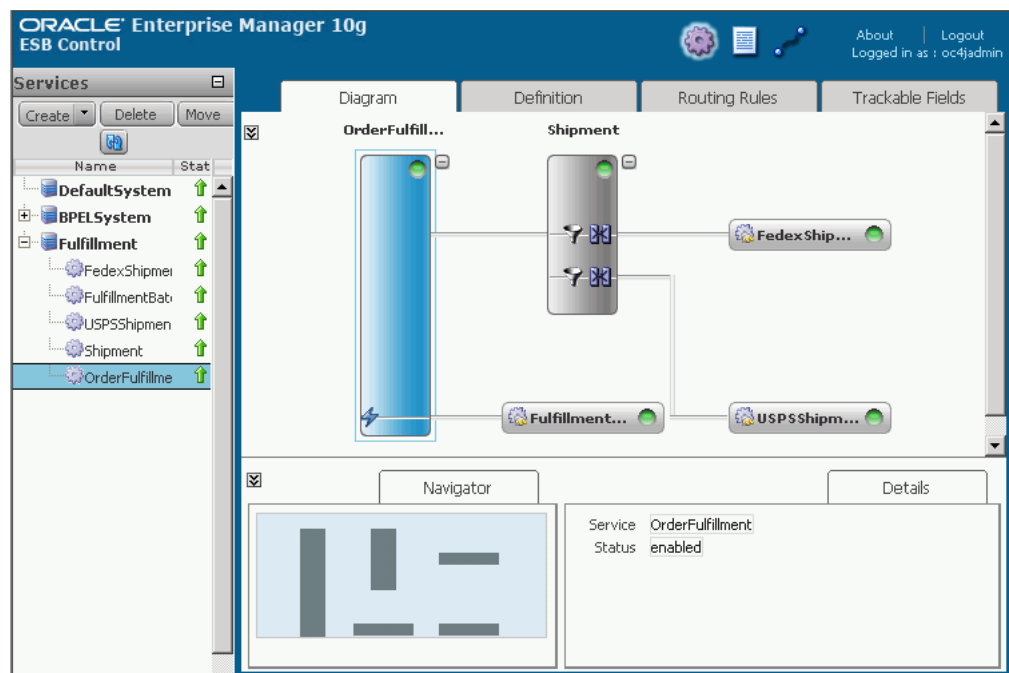
3. Review the definitions and specify values for one or more of the system definition fields if necessary.
4. Click **Apply** to save your changes.

12.4 Using the Services View to Review and Update ESB Services

The Services view of Oracle ESB Control enables you to view system and service definitions, update routing rules, define trackable fields, and view a schematic diagram of the services.

[Figure 12-2](#) provides an example of the Diagram tab of the Services view in Oracle ESB Control.

Figure 12-2 Oracle ESB Control Services View - Diagram Tab



The Services view is divided into two main regions: a services panel on the right and tabbed pages on the left. The services panel provides a navigation tree of ESB systems and the service groups and services within each, and buttons to create, delete, and move objects within the navigation tree.

The services panel appears on the left side of Oracle ESB Control. It provides buttons for creating and deleting ESB systems, groups, and services and provides a navigation tree to present these items (and their current status) after they have been created.

The tabbed panel presents details about services selected in the navigation tree. For each service, you can do any of the following:

- On the Diagram tab, view the relationship of the selected service to other objects within the ESB system, as well as statistics (latency, throughput, instances, and faults).

The selected service is represented in blue within the diagram, whereas the other services are represented in gray.

- On the Definition tab, view the definition of the selected service, enable it, disable it, or set validation of the payload at runtime. This page summarizes the contents of the WSDL file for the selected service. It consists of three regions for all services, except inbound adapter services. Inbound adapter services present the overall section only, and only a subset of the fields for that region.
 - The general region of information, which is not labeled, appears at the top of the page. The elements that are presented in this region of the page are Name, Type, Description, Invocable from an external service, Concrete WSDL, Port Type, Namespace, WSDL URL, and SOAP Endpoint URI. For inbound adapter services, only the following elements are presented: Name, Type, Description, and Inbound Adapter WSDL URL.
 - The Operations region of the page lists the names of the operations that the service WSDL describes and their types (one-way or request/response).
 - The Operation Details region of the page provides details on an operation selected in the Operations region, including validation of the payload at runtime, message payload element name, and location of the schema, if any. Which tabs are presented in this region depend on the operation definition, as follows:
 - One-way operations
If the selected operation is defined as a one-way operation (only input elements are defined for the operation in the WSDL), then only the Request tab is active.
 - Request/response operation
If the selected operation is defined as a request/response operation (both input and output elements are defined for the operation in the WSDL), then both the Request and Response tabs are active.
 - Faults
If a fault is defined for the operation (fault elements are defined for the operation in the WSDL), then the fault tab is active.
- On the Routing Rules tab, view, update, delete, and set the priority of routing rules for the selected routing service. The process is similar to the actions in the routing rules panel in Oracle JDeveloper ESB Designer. See [Section 6.10, "Specifying Routing Rules"](#).
- On the Properties tab, add, view, update, and delete endpoint properties for the selected adapter or SOAP service. This process is similar to the actions in the endpoint properties panel in Oracle JDeveloper ESB Designer. See [Section 6.6, "Adding Endpoint Properties for Adapter or SOAP Services"](#).

Note: Endpoint properties are not a production feature and are only available in Preview mode.

- On the Trackable Fields tab, define trackable fields for the selected service. See [Section 12.8, "Managing Message Instance Tracking"](#).

12.4.1 How to View and Modify Service Definitions

You can view the definition of an ESB service in the Services view of Oracle ESB Control. You can also update some of the configurations in the services Definition tab, such as enabling the validation of a payload at runtime or specifying that the service can be invoked from an external service.

To view and modify service definitions:

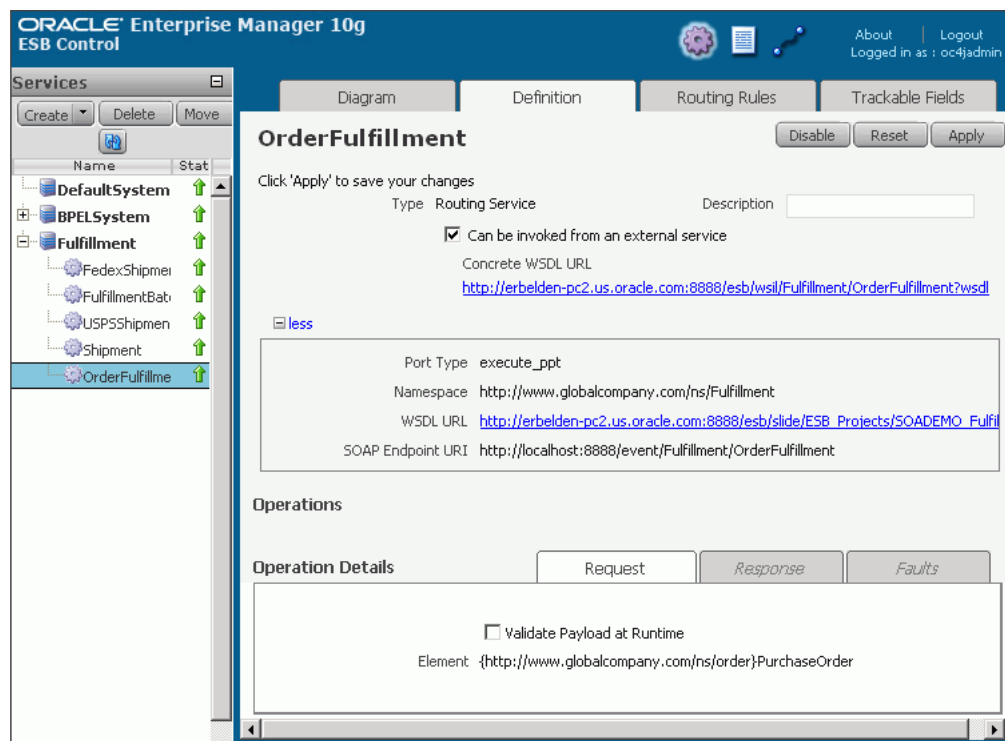
1. Click **Services** if the Services view is not currently displayed.
2. In the Services panel, navigate to the service of interest, and then select it.
3. Click the **Definition** tab.
4. Review the definitions and update one or more of the system definition fields if necessary.

For example, in the Operation Details region, click in the box next to **Validate Payload at Runtime** to enable this functionality.

5. Click **Apply** to save your changes.

[Figure 12–3](#) shows an example of the Services view with the Validate Payload at Runtime option on the Definition tab.

Figure 12–3 Services View – Definition Tab

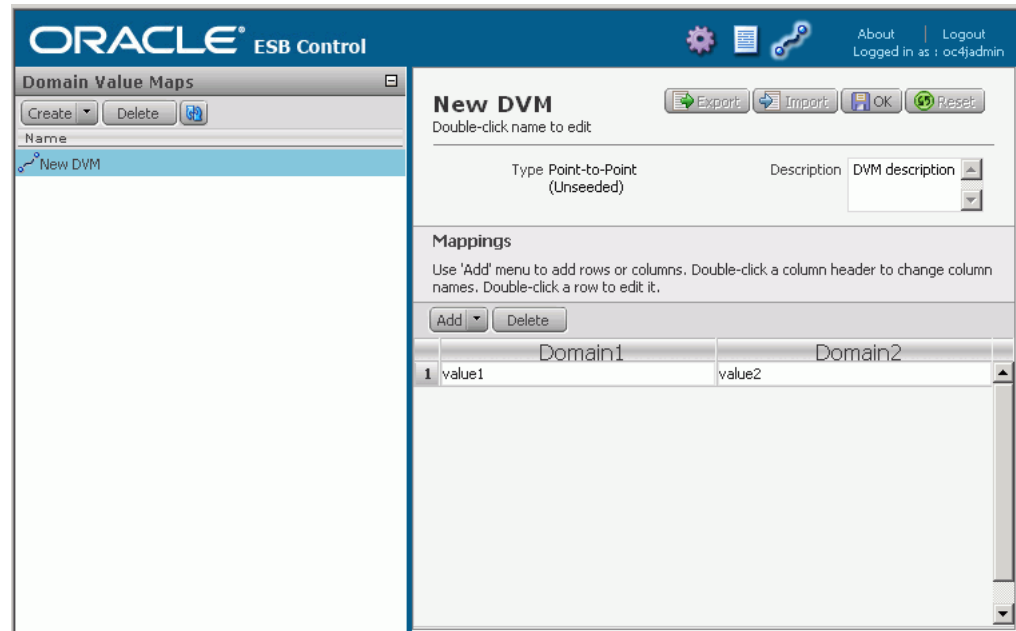


12.5 Using the Maps View to Define Domain-Value Maps

The Maps view enables you to create, update, and delete domain-value maps, as well as view, export, and import existing domain-value maps.

Figure 12–4 provides an example of the Oracle ESB Control Map view.

Figure 12–4 Oracle ESB Control Map View



Applications that you want to integrate using Oracle Enterprise Service Bus likely use different values to represent the same information. A domain-value map enables you to associate values from one application to values from another. Each domain-value map typically holds a specific category of value mappings among multiple applications. For example, one domain-value map might hold mappings for country codes and another might hold mappings for units of measurement.

For information about creating and using a domain-value map, see [Section 6.11, "Adding a Domain-Value Map to Oracle Enterprise Service Bus"](#).

For more information on the domain-value maps, see "Domain-Value Maps" in *Oracle Enterprise Service Bus Developer's Guide*.

12.6 Using the Instances View to Monitor Message Instances

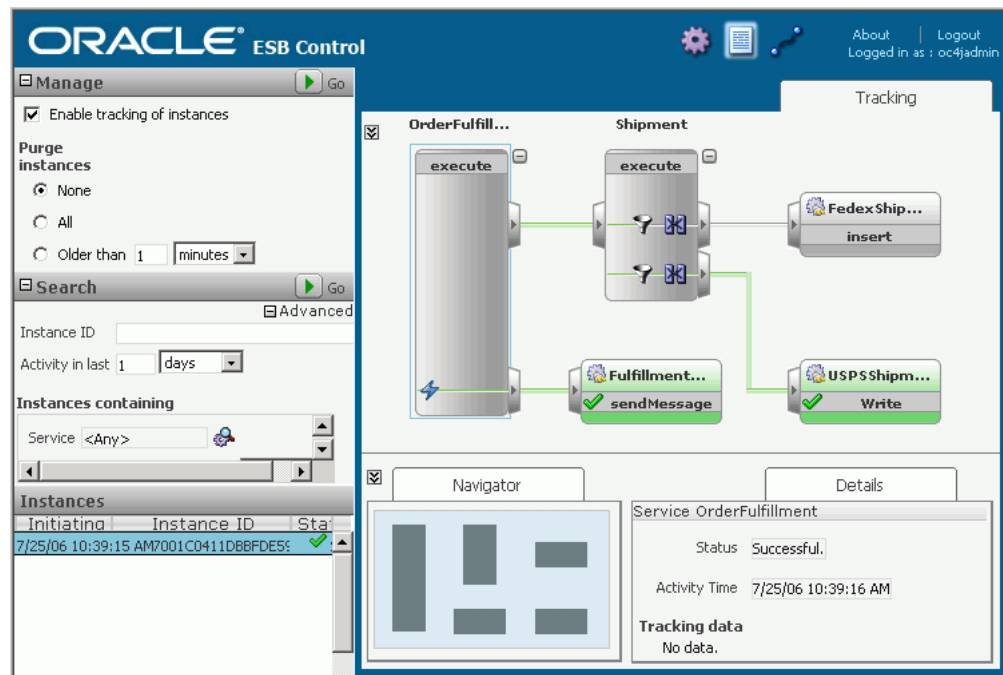
The Instances view enables you to view details about message processing across an ESB system. This view enables you to filter messages based on any of the following properties:

- The service that processed them
- The status of the messages (Any, Error, Faulted, Processing, or Completed)
- Tracking name and tracking value
- Message IDs
- Timeframe during which the message was processed

When you select a service for a given message from the Instances panel, the message instance's path through the enterprise service bus is presented in the Tracking tab diagram (and the selected service is enclosed by dotted lines in the diagram). Within the diagram, endpoints that successfully processed the message are represented in green, services where an error occurred are represented in red, and services that processed the message successfully, but were rolled back due to an error are represented in yellow. Endpoints that were not invoked in the processing of the message are represented in gray.

Figure 12–5 provides an example of the Oracle ESB Control Instances view showing successful message processing with service icons colored green.

Figure 12–5 Oracle ESB Control - Instances View



For more information about monitoring message instances, see "Tracking Message Instances Across the Enterprise Service Bus" in *Oracle Enterprise Service Bus Developer's Guide*.

12.6.1 How to View All Message Instances

You can view all the messaged instances by simply clicking the green arrow on the Search title bar on the left side of Oracle ESB Control when the Instances view is displayed.

To view all message instances:

1. At the top of Oracle ESB Control, click the **Instances** button if the Instances view is not currently displaying.
2. In the Search panel title bar, click the **Apply** icon (green arrow).

All the message instances display for the default time period specified.

12.7 Searching for Message Instances

You can search for specific message instances by specifying search criteria in the Search panel.

The search criteria are listed in [Table 12–1](#).

Table 12–1 Specifying Message Instance Search Criteria

Search Criteria	Description
Instance ID	To limit the messages to those with a specific instance ID: 1. If they are not already displaying, open the Advanced options by clicking the expand icon. (The plus (+) box next to Basic in the Instances panel.) 2. In the Instance ID field, enter the ID of the desired message instance.
Activity in last < time frame>	To limit the messages to processed in a specific timeframe: 1. If they are not already displaying, open the Advanced options by clicking the expand icon. (The plus (+) box next to Basic in the Instances panel.) 2. From the drop-down list next to the Activity in last field, select a time period. 3. In the Activity in last field, select the number of time periods specified in step 2 that you want to include. Be aware that message instances for a given timeframe may not be available if messages have been purged, as described in Section 12.8.2, "How to Purge Message Instances".
Service	To limit the messages to those processed by a particular service: 1. To the right of the Service field, click the Service Browser icon. A dialog box opens. 2. Navigate to the ESB system that contains the service of interest. 3. Select the service of interest. 4. Click Select.
Status	To limit the messages to those in a particular status, in the Status field, select one of the following: ■ Error – Select this option to limit the message instances to those for which an error occurred in processing. ■ Faulted – Select this option to limit the message instances to those for which a fault, as defined by the service WSDL, occurred. ■ Processing – Select this option to limit the message instances to those which are still being processed. ■ Completed – Select this option to limit the message instances to those for which processing across the enterprise service bus has successfully completed.
Match	Specify whether to match any or all of the criteria in the Tracking Name and Value fields.

Table 12–1 (Cont.) Specifying Message Instance Search Criteria

Search Criteria	Description
Tracking Name and Tracking Value (trackable fields)	To limit the messages to those which have a particular value in a trackable field or fields: 1. Replace <Tracking Name> with the name of a previously defined trackable field. See Section 12.9.1, "How to Define Trackable Fields" for instructions on defining trackable fields. 2. Select a logical operator from the dropdown list. 3. Replace <Tracking Value> with the value for the trackable field. 4. If you want to specify multiple trackable field/value pairs, click the plus (+) button, and then repeat steps 1 and 2 for each additional pair. 5. In the Match field, select Any if you want a message instance listed if it matches one or more of the listed tracking field/tracking value pairs specified; select All if you want only messages listed that match all of the listed tracking field/tracking value pairs. If you want to delete a trackable field/tracking value pair, click the minus (-) button that appears next to that row.

12.7.1 How to Search for Message Instances

If the Instances panel contains so many message instances that is difficult to quickly find the message instances of interest, you can search the messages with specific criteria.

To search for message instances:

1. At the top of Oracle ESB Control, click the **Instances** button if the Instances view is not currently displaying.
2. Open the Search panel, if it is not already open.
3. Specify the desired search criteria in the Search panel, as described in [Table 12–1](#).
4. In the Search panel title bar, click the **Apply** icon (green arrow).

All the message instances display for the criteria specified.

12.8 Managing Message Instance Tracking

You can manage message instance tracking with the Manage panel in the Instances view of Oracle ESB Control. In the Manage panel, you can enable or disable the tracking of message instances and purge instances.

12.8.1 How to Enable and Disable Message Instance Tracking

By default, message tracking is enabled, but it can be disabled in the Manage panel of the Instances view of Oracle ESB Control.

To enable or disable message tracking:

1. At the top of Oracle ESB Control, click the **Instances** button if the Instances view is not currently displaying.
2. Open the Manage panel if it is not already open.
3. Select or deselect **Enable tracking of instances** to enable or disable instance tracking, respectively.
4. In the Manage panel title bar, click the Apply icon (green arrow).

12.8.2 How to Purge Message Instances

By default, no message instances are purged. As a result, the list of message instances in the Instances panel can become unwieldy. You can direct the ESB Server to purge all messages or all message instances that occurred outside a specified time range.

To purge message instances:

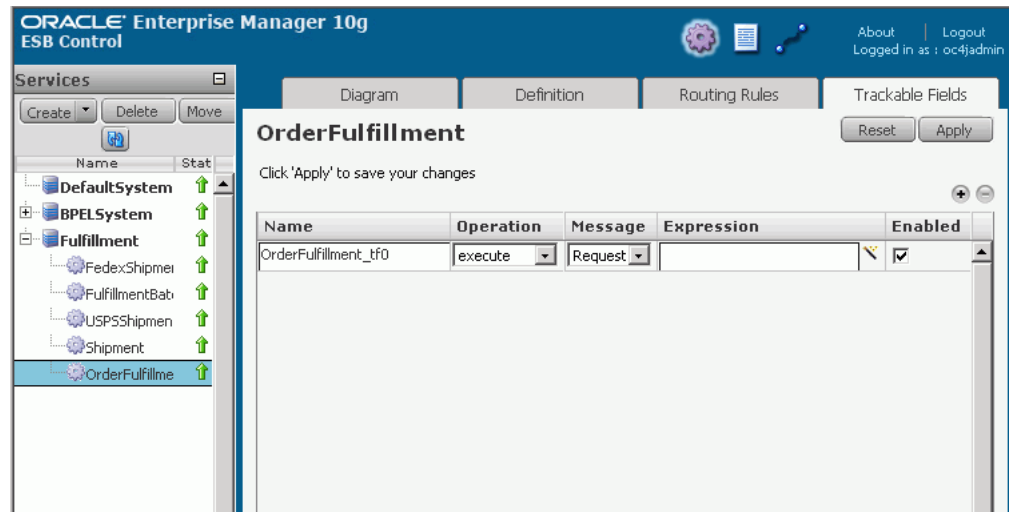
1. At the top of Oracle ESB Control, click the **Instances** button if the Instances view is not currently displaying.
2. Open the Manage panel if it is not already open.
3. Select the frequency with which you want messages purged:
 - To purge all message instances, under **Purge instances**, select **All**.
 - To purge all message instances that occurred outside a time range, under **Purge instances**, select **Older than** and indicate the time range.
4. In the Manage title bar, click the Apply icon (green arrow).

12.9 Defining Trackable Fields

Oracle ESB Control enables you to track the path particular message instances take across the Oracle Enterprise Service Bus flow and the current routing status of that message.

The mechanism by which you track messages is trackable fields. A trackable field is a name-value pair, where the name is any meaningful string that you specify and the value is an XPath expression defined for the input and output message of an entire service or an operation performed by the service.

[Figure 12–6](#) provides an example of the Trackable Fields tab of the Oracle ESB Control Services view.

Figure 12–6 Services View – Trackable Fields Tab

12.9.1 How to Define Trackable Fields

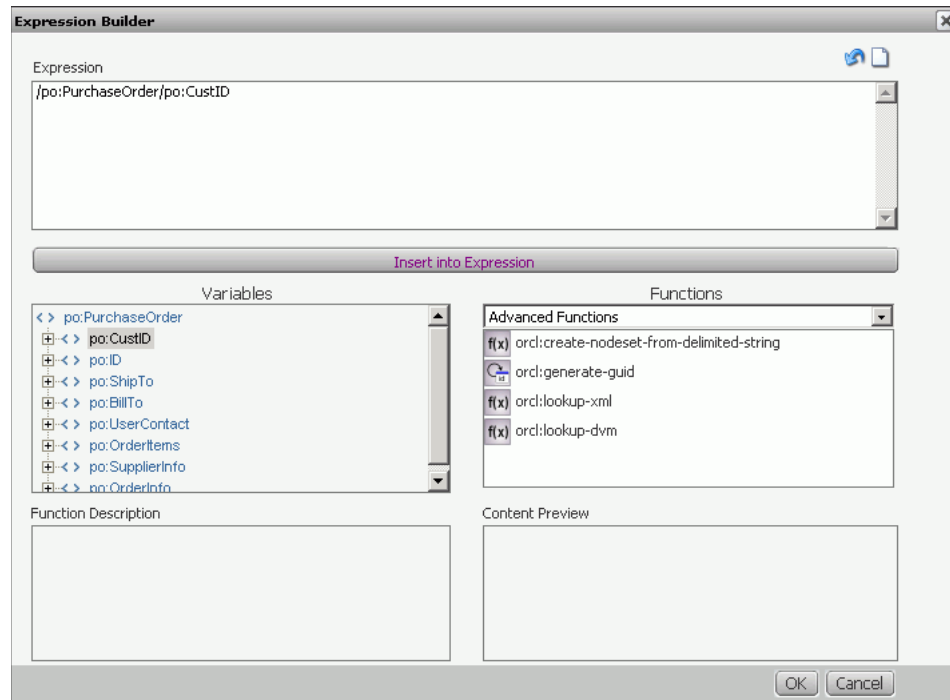
Using the Trackable Fields tab in the Services view of Oracle ESB Control, you can define one or more trackable fields for each message associated with a service and operation to keep track of the messages that the service operation processes at runtime. You can then search the message instances that the service or operation has processed using the Instances view of Oracle ESB Control.

To define a trackable field:

1. At the top of Oracle ESB Control, click the **Services** button if the Services view is not currently displaying.
2. In the **Services** panel, select the service for which you want to set trackable fields.
3. Click the **Trackable Fields** tab.
4. Click the plus (+) button to add a row to the table and enable editing for that row; double-click an existing row to update that row.
5. Enter values in the row, as follows:
 - a. In the **Name** field, enter a name for the trackable field that is unique across the ESB system in which the service exists.
 - b. In the **Operation** field, select the operation on which you want to define the trackable field.
 - c. In the **Message** field, select the type of message you want to track. Possible values are: Request, Response, Fault.
 - d. In the **Expression** field, enter the XPath expression that identifies the field that you want to track.

Click the wand icon to the right of the **Expression** field to open the Expression Builder, which assists you in building the XPath expression.

[Figure 12–7](#) provides an example of Expression Builder used to identify the field you want to track.

Figure 12–7 Expression Builder

- e. In the **Enabled** field, select the check box to enable the specified trackable field; deselect the checkbox to disable it.

If you disable the trackable field, the trackable field information is not logged and is not available in the Instances view of Oracle ESB Control.
 - f. Click **OK** when finished.
6. Click **Apply** to save the changes you have made; click **Reset** to return to the values that were presented before you began editing fields.

12.9.2 How to Enable or Disable Trackable Fields

You can enable or disable trackable fields in the Services view of Oracle ESB Control.

To disable or enable a trackable field:

1. At the top of Oracle ESB Control, click the **Services** button if the Services view is not currently displaying.
2. Click the **Trackable Fields** tab.
3. Double-click the row that contains the trackable field that you want to enable or disable.
4. In the **Enabled** column, select the checkbox to enable the trackable field or deselect it to disable the trackable field.
5. Click **Apply**.

12.9.3 What Happens When You Define a Trackable Field

Oracle Enterprise Service Bus monitors the expression for the field during message instance processing.

12.9.4 What Happens at Runtime

At runtime, the Details tab in the Instance view displays tracking data for this service based on the criteria set up in the expression field of the trackable field.

12.10 Viewing Log Files

You can view logs for all the OC4J components running on an application server with Oracle Enterprise Manager Application Server Control. Oracle Application Server components generate log files containing messages that record all types of events, including startup and shutdown information, errors, warning messages, access information on HTTP requests, and additional information.

12.10.1 How to View Log Files

You can view ESB log files through the Log Files page of the Oracle Enterprise Manager Application Server Control.

To view log files using Oracle Enterprise Manager:

1. Open Oracle Enterprise Manager Application Server Control for your system.
2. Click an application server to display the home page of the server.
3. Click **Logs** in the upper right corner to display the Log Files page.
4. Expand **Components** > **OC4J** > *name-of-application server*.

The Console appears with output similar to the screenshot in [Figure 12–8](#).

Figure 12–8 Oracle Application Server Control – OC4J Log Files

ORACLE Enterprise Manager 10g
Application Server Control

Setup Logs Help Logout

Cluster Topology > Application Server: soademo.erbelden-pc2.us.oracle.com >

Log Files

Page Refreshed Jun 27, 2006 1:28:40 PM PDT

View All Logs

Select log files and... Search

Select All | Select None | Expand All | Collapse All

Select Item	Log Type	Modified	Size (bytes)	View	Search
☐ Components					
☐ Enterprise Manager					
☐ OC4J					
☐ home:1					
☐ Application ACMECustomerSOAPService					
☐ Application CustomerService					
☐ Application SOADEMO-CREDITSERVICE-CreditService-WS					
☐ Application SOADEMO-RAPIDSERVICE-RapidService-WS					

- Expand the items in the Select list until you locate a specific log file, and then click **View**.

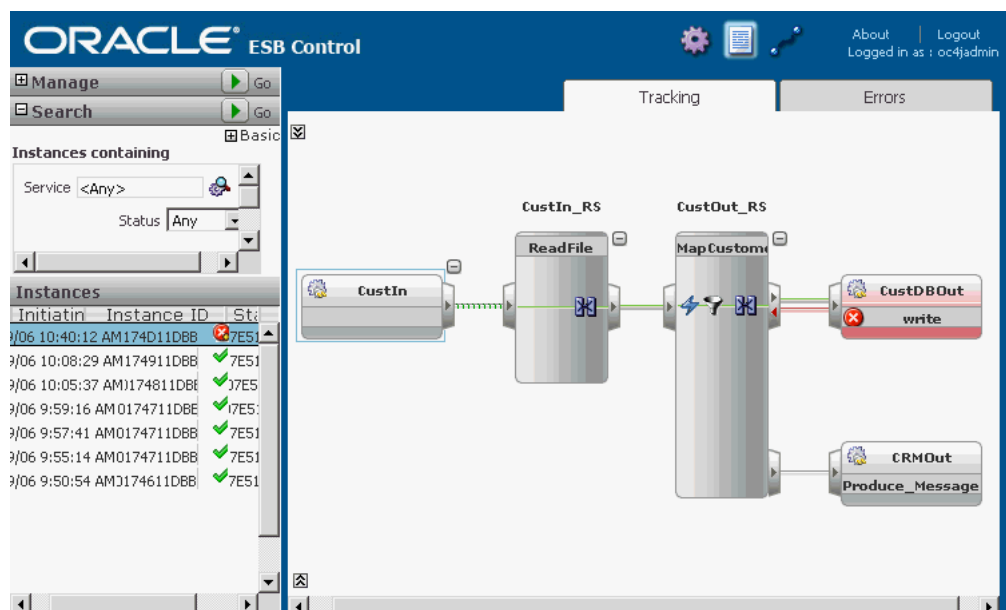
For example, expand the Application esb-dt, Application esb-rt, and Diagnostic Logs items. Click the icon in the View column to display the log file text.

For more information about the log files of Oracle Application Server components, see "Managing Log Files" in *Oracle Application Server Administrator's Guide*.

12.11 Error Handling in Message Instance Processing

When an error occurs in Oracle Enterprise Service Bus processing, the error is noted by visual cues, such as icon and color changes, in Oracle ESB Control. A red icon identifies an error condition, as shown in [Figure 12–9](#).

Figure 12–9 *Instances View – Tracking Tab with Error*



In addition to the visual cues in Oracle ESB Control, you can set up notifications by email, fax, or phone when errors occur. See [Section 12.3.1, "How to View and Modify System Definitions"](#).

Error handling in Oracle Enterprise Service Bus involves several types of errors that can occur in transaction processing:

- **Rejection:** A validation error in the message that must be fixed by the user
- **Retry:** A temporary loss of a service impacting the routing of the message, but that is usually resolved in a relatively short time period
- **Fatal:** A disabled or deleted service or system, out of memory condition, or other serious problem that results in a serious error requiring attention by a system administrator

Error handling is processed differently whether asynchronous and synchronous execution is specified for a routing service. In general, errors during synchronous execution cannot be retried and errors during asynchronous execution can be resubmitted.

- For synchronous execution, the transaction is rolled back and an error notification is returned to the adapter that initiated the processing. The calling adapter is expected to handle the error and resubmission process. See [Section 12.11.1, "About Adapter Error Handling"](#).

For more information about adapters, including a discussion of managing errors, see *Oracle Application Server Adapter Concepts*.

- For asynchronous execution, the user can resubmit the transaction after the error condition has been resolved. See [Section 12.11.2, "How to Resubmit Messages on Errors"](#).

For more information about error handling in Oracle Enterprise Service Bus, see "Error Handling" in *Oracle Enterprise Service Bus Developer's Guide*.

12.11.1 About Adapter Error Handling

An adapter handles exceptions and faults using the default error handling process of the adapter.

- By default, an adapter retries the message three times at five-second intervals for an error condition. The retry count and interval can be specified in the endpoint properties of the adapter service. For information about endpoint properties, see [Section 6.6, "Adding Endpoint Properties for Adapter or SOAP Services"](#).
- If an inbound adapter fails to invoke a routing service for a certain number of consecutive times, it marks itself broken and disables itself. Oracle ESB Control displays this event source with a special icon to visually represent its disabled state. You can enable the adapter service.
- If the next service that the inbound adapter invokes does not exist, perhaps because it has been deleted or is not enabled, then the inbound adapter processor disables itself and marks itself broken.
- If a subscription fails a certain number of consecutive times, the service notifies the repository to mark it as in a broken state. The dispatcher does not dispatch this subscription after it is marked as broken.

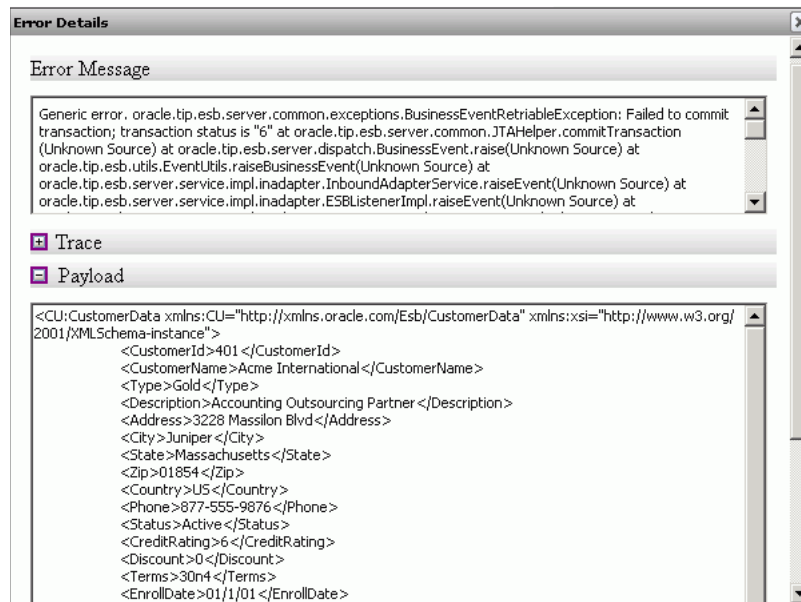
12.11.2 How to Resubmit Messages on Errors

When an error occurs while a message is being processed, it is indicated by a red icon next to the routing service in Oracle ESB Control, as shown in [Figure 12-9](#).

In some situations, a message instance can be resubmitted. If a routing rule has been set to asynchronous execution, you can resubmit a message instance after fixing the error condition.

You can view error message, trace, and payload details by clicking the **Error Details** icon under the Message column on the Errors tab, as shown in [Figure 12-11](#).

[Figure 12-10](#) provides an example of the Error Details dialog.

Figure 12–10 Error Details Dialog

After you review the error details and fix the error condition, you can resubmit the message to the invoking service.

To resubmit a message on error:

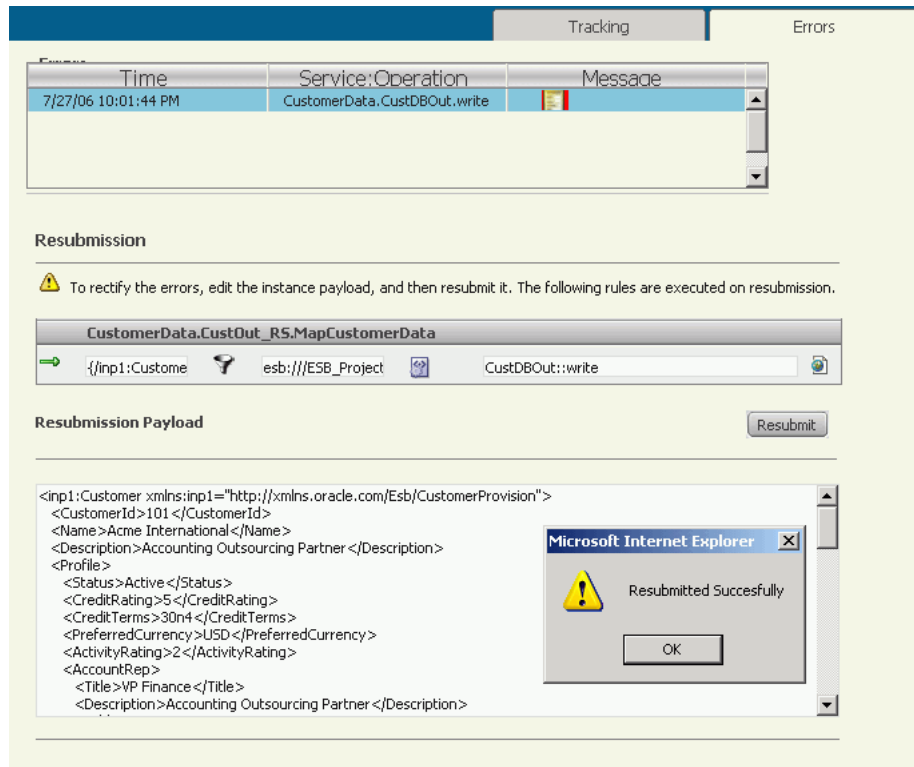
1. At the top of Oracle ESB Control, click the **Instances** icon to display the message instance processing.
2. In the Instances panel of the Instances view, click the message instance where the error occurred.

The Tracking tab appears, similar to [Figure 12–9](#).

3. Click the **Error** tab to display the error information.
4. Click the **Error Details** icon under the Message column to view error message, trace, and payload details about the error condition in the Errors Detail dialog.
5. Click **OK** to close the Errors Detail dialog after reviewing the error message details.
6. Correct the error condition, and then click **Resubmit** in the Error tab.

For example, edit the message payload in the Resubmission Payload window if is incorrect and then click **Resubmit**.

[Figure 12–11](#) provides an example of the resubmitted message instance.

Figure 12–11 *Instances View - Error Tab*

12.11.3 What Happens When You Resubmit Messages on Errors

When you resubmit a message, ESB begins processing the message again. If the error condition has been corrected, the message processing should be successful.

Monitoring Oracle BPEL Process Manager

This chapter describes how to monitor and manage BPEL processes and domains using Oracle BPEL Control.

This chapter includes the following sections:

- [Section 13.1, "Introduction to Oracle BPEL Control"](#)
- [Section 13.2, "Components of Oracle BPEL Control"](#)
- [Section 13.3, "Using the Dashboard Tab to Monitor Deployed, Running, and Completed Processes"](#)
- [Section 13.4, "Using the BPEL Processes Tab to Manage a Process Life Cycle"](#)
- [Section 13.5, "Using the Instances Tab to Monitor Process Instances"](#)
- [Section 13.6, "Using the Activities Tab to Monitor Process Activities"](#)
- [Section 13.7, "Managing BPEL Domains"](#)

13.1 Introduction to Oracle BPEL Control

Oracle BPEL Control is a browser-based tool for managing the life cycle of a BPEL process. You can use it to view currently running BPEL processes to determine the success of the processing and then correct any problems in Oracle BPEL Process Manager.

Oracle BPEL Control is also useful for deciding when a process is ready for production mode. For example, assume that you compiled and deployed version 1.0 of your BPEL process and subsequently a newer version, version 2.0. After testing and verification of your BPEL process, you can use Oracle BPEL Control to mark one of the versions as the default. Then new instances are started using this definition.

Read this chapter to understand:

- How to monitor deployed, running, and completed processes
- How to manage a process life cycle and initiate processes
- How to monitor process instances
- How to monitor process activities
- How to change domains

13.2 Components of Oracle BPEL Control

Oracle BPEL Control provides the following components:

- **Dashboard** tab: Displays deployed, running, and completed processes
- **BPEL Processes** tab: Displays the process life cycle and enables you to provide input to initiate a process
- **Instances** tab: Displays process instances
- **Activities** tab: Displays process activities

You can also change BPEL domains from Oracle BPEL Control

13.2.1 How to Log In to Oracle BPEL Control

Use Internet Explorer 6.0 or Mozilla Firefox 1.0.4 to access Oracle BPEL Control. Oracle BPEL Server must be running to access Oracle BPEL Control.

To log in to Oracle BPEL Control:

1. Open a Web browser.
2. Go to the following URL:

`http://localhost:portnumber/BPELConsole`

The *localhost* and *portnumber* used at installation are noted in `bpelsetupinfo.txt`, at

`SOA_Oracle_Home\install\`

Or select **Start > All Programs > Oracle - Oracle - soademo > Oracle BPEL Process Manager > BPEL Control**

3. Type `oc4jadmin` in the **Username** field and `welcome1` in the **Password** field.

13.3 Using the Dashboard Tab to Monitor Deployed, Running, and Completed Processes

The **Dashboard** tab displays the currently deployed BPEL processes and instances of BPEL processes that are currently running (in-flight) and that have recently completed. An asterisk identifies the version that is the default process. [Figure 13–1](#) shows the **Dashboard** tab for the **SelectManufacturer** process in the SOA Order Booking application.

Figure 13–1 Oracle BPEL Control: The Dashboard Tab

BPEL Control			
Logged to domain: default			
Dashboard BPEL Processes Instances Activities			
Deployed BPEL Processes		In-Flight BPEL Process Instances	
Name	Instance	BPEL Process	Last Modified ↑
SelectManufacturer			

[Figure 13–2](#) shows the **Dashboard** tab for several BPEL processes with multiple versions.

Figure 13–2 Oracle BPEL Control: The Dashboard Tab Showing Multiple Versions

Dashboard			
BPEL Processes			
Instances			
Activities			
Deployed BPEL Processes		In-Flight BPEL Process Instances 1 - 3	
Name	Instance	BPEL Process	Last Modified ↑
BatchOrderProcessing	34 : Instance #34 of OrderBooking	OrderBooking (v. 1.5)	7/23/06 11:30:56 AM
CreditRatingService	30 : Instance #30 of OrderBooking	OrderBooking (v. 1.4)	7/23/06 11:06:07 AM
FulfillOrder	32 : Instance #32 of SelectManufacturing	SelectManufacturing (v. 1.0)	7/23/06 11:05:29 AM
OrderApproval (v. 1.0)			
OrderApproval (v. v2006_07_25__37316) ★			
OrderBooking (v. 1.0)			
OrderBooking (v. 1.1)			
OrderBooking (v. 1.2)			
OrderBooking (v. 1.3)			
OrderBooking (v. 1.4)			
OrderBooking (v. 1.5) ★			
POAcknowledge			
RapidDistributors			
ReportsDemo (v. 1.0)			
ReportsDemo (v. 2.0)			
ReportsDemo (v. 3.0) ★			
SelectManufacturing			
TaskActionHandler			
TaskManager			
Recently Completed BPEL Process Instances (More...)			
	✓ 149 : Instance #149 of ReportsDemo	ReportsDemo (v. 3.0)	7/24/06 7:11:14 PM
	✓ 148 : Instance #148 of ReportsDemo	ReportsDemo (v. 3.0)	7/24/06 7:05:50 PM
	✓ 147 : Instance #147 of ReportsDemo	ReportsDemo (v. 3.0)	7/24/06 7:05:44 PM

13.3.1 How to Monitor Deployed, Running, and Completed Processes

Use the **Dashboard** tab to monitor deployed, running, and completed processes. If a running process is awaiting input, use the **Initiate** page to provide it.

To monitor deployed, running, and completed processes, and to initiate processes:

1. Click the **Dashboard** tab.
2. Click a deployed BPEL process in the **Name** column to access a page for creating an instance and testing your process.

If you clicked the name of a running process, the **Initiate** page is displayed.

3. For a running process awaiting input, provide the input needed to complete the process, as shown in [Figure 13–3](#).

Figure 13–3 Oracle BPEL Control: The Initiate Page of the BPEL Processes Tab

BPEL Control Logged to domain: **default**

Dashboard **BPEL Processes** **Instances** **Activities**

BPEL Process: **SelectManufacturer** Version: 1.0 Lifecycle: Active
 Statistics: [0 Open Instances](#) | [0 Closed Instances](#)

[Manage](#) **Initiate** [Descriptor](#) [WSDL](#) [Sensors](#) [Source](#) [Test Suites](#) [Reports](#)

Testing this BPEL Process [Through SOAP](#) | [Through Java Delivery API](#)

Initiating a test instance
 To create a new 'test' instance of this BPEL Process, fill this form and click on the 'Post XML Message' button.

Operation: **processRequestQuote** ☒ HTML Form ☐ XML Source

☒ WS-Security ☐ Include In Header

☒ WS-Addressing ☐ Include In Header

☒ parameters

☒ param0 ☒ Include In Message

itemId xsd:string ☒ Include In Message
 quantity xsd:long

Note: XML source view contents will not be reflected in the HTML form view

☐ Save Test
☐ Perform stress test

Post XML Message

4. Click Post XML Message.

13.4 Using the BPEL Processes Tab to Manage a Process Life Cycle

The **BPEL Processes** tab displays process life cycles and states, including the version numbers of BPEL processes that are currently active. An asterisk identifies the default process.

Figure 13–4 shows the **BPEL Processes** tab for the **SelectManufacturer** process in the SOA Order Booking application.

Figure 13–4 Oracle BPEL Control: The BPEL Processes Tab for the SelectManufacturer Process

The screenshot shows the Oracle BPEL Control interface. At the top, there's a header bar with "BPEL Control" on the left and "Logged to domain: default" on the right. Below this is a navigation bar with four tabs: "Dashboard", "BPEL Processes" (which is selected), "Instances", and "Activities".

The main content area is divided into two sections. On the left is the "Search Processes" section, which includes a "Process Name" text input field, "State" and "Lifecycle" dropdown menus (both set to "Any"), and a "Go" button. Below this is the "Related Tasks" section with a list of links: "Clear WSDL Cache", "Deploy New Process", "Perform Manual Recovery", "Refresh Alarm Table", and "View Process Log".

On the right is the "Deployed Processes" section. It features a table with the following columns: "BPEL Process", "Lifecycle", "State", "Open Instances", and "Closed Instances". The table contains one row for the "SelectManufacturer (v. 1.0)" process, which has a lifecycle of "Active" and a state of "On", with 0 open and 0 closed instances. Below the table are links for "Check All" and "Clear All", and a "Bulk Update" button.

Use the following links on the **BPEL Processes** tab to do related tasks:

- **Clear WSDL Cache:** Clears the cache for all WSDLs of the selected domain
- **Deploy New Process:** Deploys BPEL processes from Oracle BPEL Control instead of using Oracle JDeveloper
- **Perform Manual Recovery:** Performs a manual recovery of messages. For example, if you are using the file adapter and your system server crashes while inbound messages are being processed, you can manually recovery undelivered messages when the server is restarted. If a file has ten messages and the server crashes after three messages have been processed, then the fourth message is not delivered. When the server restarts and begins processing with message five (the offset of the last successfully rejected message), you can manually recover the fourth message.
- **Refresh Alarm Table:** Refreshes the alarm table for the selected domain. This registers all pending wait/onAlarm activities with the system.
- **View Process Log:** Shows the events of all BPEL processes in the selected domain (for example, when a process was compiled, undeployed, or marked as the default instance)

13.4.1 How to Manage a Process Life Cycle

Use the **BPEL Processes** tab to manage a process life cycle.

To manage the process life cycle:

1. Click the **BPEL Processes** tab.
2. Click a specific process in the **BPEL Process** list.
3. Use the **Manage** tab to manage the life cycle and state of the BPEL process.

See [Section 13.4.2, "How to Manage a Specific Process"](#) for more information.

13.4.2 How to Manage a Specific Process

Figure 13–5 shows the **BPEL Processes** tab after you select the **SelectManufacturer** process.

Figure 13–5 Oracle BPEL Control: The Manage Page of the BPEL Processes Tab

BPEL Control Logged to domain: default

Dashboard **BPEL Processes** **Instances** **Activities**

BPEL Process: SelectManufacturer Version: 1.0 Lifecycle: Active
 Statistics: [0 Open Instances](#) | [0 Closed Instances](#)

Manage [Initiate](#) [Descriptor](#) [WSDL](#) [Sensors](#) [Source](#) [Test Suites](#) [Reports](#)

Managing this BPEL Process

Process Lifecycle:
 When the process lifecycle is **active** instances may be instantiated from the process; when it is **retired**, no new instances may be instantiated but existing instances are permitted to complete normally.

☒ Active ☐ Retired

Process State:
 The process state controls overall access to the process. When the state is **on** instances may be instantiated and accessed; when it is **off** no new instances may be instantiated and access to existing instances and activities belonging to the process will be denied.

☒ On ☐ Off

Default Revision:
 The default revision is selected if an invocation message does not specify the revision tag of the process to invoke. *This revision is currently selected as the default revision.*

Undeploy this BPEL Process
 Any open instances belonging to this process will be marked as stale upon undeployment. Users are advised to change the process lifecycle to **retired** and complete any remaining instances before proceeding.

Use the following links on the **BPEL Processes** tab to manage the **SelectManufacturer** process:

- **Manage:** (Displayed in the figure.) For each BPEL process, Oracle BPEL Control provides the following status indicators.
 - **Process Life Cycle:** A process life cycle can be active or retired. If the process life cycle is retired, you cannot create a new instance.
 - **Process State:** A process state can be on or off. If the process state is off, you cannot access instances or create new ones.
 - **Default Revision:** The designated process and revision that is instantiated when a new request comes in. There is only one default process, identified by an asterisk. If you retire a default process, the default does not change to another process. The retired process remains the default. You must explicitly select a new default process.

Designating a process as the default works as follows: If you deploy version 1.0 of a BPEL process, it is displayed as the default process in Oracle BPEL Control. If you deploy version 2.0 of that process, it is displayed as the default. If you redeploy version 1.0 of the same process, it is again displayed as the default.

- **Undeploy the BPEL Process:** Undeploy removes all traces of the process from the system. You cannot view previously completed processes. Instances belonging to this process are usually purged before undeploying a process. Undeploying the only version of a process (which is also the default) results in the complete removal of this process.
- **Initiate:** Use this page to run processes from the **BPEL Processes** tab. This is the same window that displays when you click a process in the **Deployed BPEL Processes** list of the **Dashboard** tab.
- **Descriptor:** Use this page to view and change deployment descriptor `bpel.xml` file properties of a BPEL process at runtime. This prevents you from having to reset these properties during design time and redeploy the BPEL process. See *Oracle BPEL Process Manager Developer's Guide* for information about deployment descriptor properties.
- **WSDL:** Use this page to view the WSDL file contents for a process.
- **Sensors:** Use this page to view the fault, activity, and variable sensor data of a process.
- **Source:** Use this page to view the BPEL file contents of a process
- **Test Suites:** Use this page to run the test cases of a test suite on a BPEL process instance and view XML document reports. By default, report results are formatted as JUnit XML test results. See *Oracle BPEL Process Manager Developer's Guide* for more information about test suites.
- **Reports:** Use this page to create reports in Oracle BPEL Control that enable you to
 - Receive an overall view of business process instance performance
 - Analyze data for the BPEL process instances and make critical decisions
 - Analyze data of the activities that constitute a business process
 - Identify and debug faults and take appropriate corrective actions

See *Oracle BPEL Process Manager Developer's Guide* for more information on Oracle BPEL Control reports.

13.5 Using the Instances Tab to Monitor Process Instances

The **Instances** tab displays BPEL process instances. [Figure 13–6](#) shows the **Instances** tab.

Figure 13–6 Oracle BPEL Control: The Instances Tab

Dashboard

BPEL Processes

Instances

Activities

Locate Instances

INSTANCE ID#

TITLE

PRIORITY

BPEL PROCESS

All Processes

CREATION DATE

All Times

STATE

All States

Go

List of BPEL Process Instances 1 - 5

Instance	BPEL Process	Last Modified ↑
✓ 3 : Instance #3 of OrderBooking	OrderBooking (v. 1.1)	2005-04-12 19:07:53.5
✓ 5 : Instance #5 of RapidDistributors	RapidDistributors (v. 1.0)	2005-04-12 19:07:53.04
✓ 4 : Instance #4 of CreditRatingService	CreditRatingService (v. 1.0)	2005-04-12 19:07:30.96
✓ 1 : Instance #1 of OrderBooking	OrderBooking (v. 1.0)	2005-04-12 19:05:07.01
✓ 2 : Instance #2 of CreditRatingService	CreditRatingService (v. 1.0)	2005-04-12 19:05:06.75

Related Tasks

Archive Instances

Purge All Instances

13.5.1 How to Monitor Process Instances

Use the **Instances** tab to monitor BPEL processes.

To monitor process instances:

1. Click the **Instances**.
2. Click an instance in the **Instance** column.

From the window that appears, you can perform the following tasks:

- View the state of the instance (for example, completed, active, or faulted)
- Delete the instance
- Click **Flow** to view a visual representation of the history of the activities in this instance
- Click **Audit** to view an audit trail of this instance
- Click **Debug** to view the BPEL Debugger, which takes the BPEL source code that implements this process and matches it against the state of this particular instance. Points in the code where execution is currently paused are highlighted in yellow (for example, the process is currently waiting for a loan service to call back with a loan offer).
- Click **Interactions** to view details about the activities in this instance
- Click **Sensor Values** to view the results of any activity, fault, or variable sensors you created in this instance

- Click **Test** to save an instance as a test case. You can then import it into an Oracle JDeveloper project.

Note: The **Flow** and **Audit** links do not provide details about the actions or states of any header variables you defined during design time. As a workaround, use the **Debug** link to view header variable details.

See *Oracle BPEL Process Manager Developer's Guide* for more information on the following topics:

- Using sensors
- Testing BPEL processes

13.6 Using the Activities Tab to Monitor Process Activities

The **Activities** tab displays the status of activities in the deployed BPEL process instance. [Figure 13–7](#) shows the **Activities** tab.

Figure 13–7 Oracle BPEL Control: The Activities Tab

Use the **Activity State** list to sort by open, completed, canceled, or stale states, or to view all states.

13.7 Managing BPEL Domains

BPEL processes (specifically, the suitcase JAR file) are deployed to domains. A BPEL domain allows a developer or administrator to partition a single instance of Oracle BPEL Process Manager into multiple virtual BPEL sections. A BPEL domain is identified by an ID and protected by a password. When Oracle BPEL Process Manager is installed, an initial domain named `default` is created. The initial password for accessing the default domain or any domain you create is the same as that specified for the `oc4jadmin` user during installation.

Here are some examples of how to use BPEL domains:

- Partition a single Oracle BPEL Process Manager instance into a multideveloper environment. In this case, the domain ID typically identifies the developer owning that domain.
- Partition a single Oracle BPEL Process Manager instance into both a development and QA environment. In this case, the domain IDs can be `test` and `qa`.

- Partition a single Oracle BPEL Process Manager instance into an environment used by multiple departments or partners. In these cases, the domain IDs are the names of the departments or partners.

13.7.1 How to Create BPEL Domains

The Oracle BPEL Control window includes the **Domain** list to display a list of available domains (in the upper right corner), as shown in [Figure 13–8](#).

Figure 13–8 Oracle BPEL Control: List of Domains



The domain that you are currently logged in to is always displayed.

You can create additional BPEL domains using Oracle BPEL Admin Console. The domains you create are then listed in Oracle BPEL Control.

To create BPEL domains:

1. On the Oracle BPEL Control login page, select **Go to BPEL Admin**.
2. Log in using the `oc4jadmin` username and password.
3. Click the **BPEL Domains** tab.
4. Click **Create New BPEL Domain**.
5. Follow the on-screen instructions to create a new domain with an ID.
6. Return to Oracle JDeveloper.
7. Right-click a process.
8. Select **Deploy > connection_name > Refresh**.
9. Select **Deploy > connection_name > Deploy to domain_name domain**, where *domain_name* is the ID you entered in Step 5.
10. Log in to Oracle BPEL Control.
11. Select the new domain name from the drop-down list in the upper right corner of Oracle BPEL Control.

The process you deployed in Step 9 displays in the **Dashboard** tab.

See *Oracle BPEL Process Manager Administrator's Guide* for more information about Oracle BPEL Admin Console.

Monitoring Security

This chapter describes how to assess and monitor security in an SOA application.

This chapter includes the following sections:

- [Section 14.1, "Introduction"](#)
- [Section 14.2, "Monitoring Security"](#)

14.1 Introduction

A service oriented architecture requires the ability to secure access to the services and processes that participate in the business flow. [Chapter 10, "Securing the System"](#) described how services may be secured with Oracle Web Services Manager agents and gateways.

A related requirement is the logging of message traffic throughout the process flow. Key drivers for this include regulatory concerns, compliance, and monitoring of Service Level Agreements (SLAs). If this requirement is not addressed, it becomes difficult to track unauthorized transactions and achieve compliance goals.

Oracle Web Services Manager (Oracle WSM) provides tools to monitor the status and operation of the managed services, enabling an organization to attain operational best practices across its service oriented architecture.

Read this chapter to understand:

- How Oracle Web Services Manager monitors service operation
- The metrics and statistics that are available to a security administrator
- The use of alarms and notifications when processing metrics
- The management features you can use to control access to system features

14.2 Monitoring Security

Oracle Web Services Manager provides extensive features to enable you to Oracle Web Services Manager monitor the operational health of the components (gateways and agents) that you set up to secure your web services.

This section surveys the monitoring capabilities with these topics:

- [Section 14.2.1, "The Oracle Web Services Manager Monitor"](#)
- [Section 14.2.2, "Execution Metrics"](#)
- [Section 14.2.3, "Security Statistics"](#)
- [Section 14.2.4, "Service Statistics"](#)
- [Section 14.2.5, "Alarms"](#)
- [Section 14.2.6, "Role Management"](#)

See Also: *Oracle Web Services Manager User and Administrator Guide*
for details about Oracle Web Services Manager Monitor

14.2.1 The Oracle Web Services Manager Monitor

The Oracle Web Services Manager Monitor collects Oracle Web Services Manager operational status and other service execution metrics from various sources. An Oracle Web Services Manager gateway sends metrics to the Monitor for each Web service managed by the gateway. Agents send metrics to the Monitor from the point where they are embedded, whether on the client side or the server side. The Monitor collects these metrics and makes them available to the administrator so that the performance and security of the system can be actively managed.

To access the monitor:

1. Point your browser to the Web Services Manager Control Console and log in using your single sign-on user name and password.

The Web Services Manager Control Console is accessed with a URL of the form:

`http://<hostname>:port_number/ccore`

For example:

`http://itapps.globalcompany.com:8888/ccore`

2. Click **Operational Management** in the left navigation panel of the Console to access the Monitor.

14.2.2 Execution Metrics

You can view Oracle Web Services Manager execution metrics in the Snapshot menu under **Operational Management**.

Oracle WSM Monitor provides these broad measurements of the managed services:

- Snapshot - The snapshot provides a visual dashboard-style display of security statistics, including:
 - the average failure rates for policy and service execution steps for a gateway or agent, for a selected service and across services
 - the average latency for policy and service execution steps for a gateway or agent, for a selected service and across services
 - failure rates for authentication and authorization steps of a policy for a selected service and across services
- Comparisons of performance metrics against SLA (Service Level Agreement) conformance limits that you can configure

- Execution details for a selected component and service
- Message logs for service requests and responses. For failures, you can drill down to view error messages.
- Flows, which are groups of Web service invocations within a specific context, such as a chain of back-end application services that fulfill an order-processing request

14.2.3 Security Statistics

Oracle WSM Monitor provides Access Control statistics showing the distribution of access over all service execution attempts, and the distribution of granted requests, authentication failures, and authorization failures across all access attempts.

You can view these statistics by clicking **Security Statistics** under **Operational Management**.

14.2.4 Service Statistics

Oracle WSM Monitor provides these statistics about the managed services:

- Latency Variance - Displays a graph showing minimum and maximum latency metrics, by individual service or across all services.
- Traffic Analysis - Displays the number of bytes and the messages per service, by individual service or across all services.

You can view these statistics by clicking **Service Statistics** under **Operational Management**.

14.2.5 Alarms

Oracle Web Services Manager Oracle WSM Monitor lets you Oracle Web Services Manager create alarms that are used in processing the collected metrics. You create an alarm rule to define what constitutes the alarm; for example, you could create an alarm if average latency drops below a certain level. You then create a rule action to be executed when the rule is triggered. For example, a typical action could be an alert that is sent to key personnel.

You can view and manage alarms by clicking **Alarms** under **Operational Management**.

You can search for alarms by a combination of criteria including component, alarm type, time range, and severity.

14.2.6 Role Management

To control access to Oracle WSM features and Oracle Web Services Manager permissions for Oracle WSM operations, you can map user groups defined in your environment to Oracle WSM administrative roles. Typical roles include Domain Administrator (responsible for the day-to-day operations and management of an Oracle WSM site), Component Administrator (responsible for managing services associated with a specific gateway or agent), Service Administrator (responsible for a specific service), and Service Support (able to view a specific service), among others.

Users logging in to Oracle WSM Web Services Manager Control are restricted in the operations they are able to perform based on their roles. Certain roles are also able to delegate administrator functions to other users.

Access control tools are available under the **Administration** menu.

Monitoring SOA Applications with Enterprise Manager

This chapter describes how to assess and monitor the performance of an SOA application using Oracle Enterprise Manager 10g Application Server Control.

This chapter includes the following sections:

- [Section 15.1, "Introduction to Monitoring the OC4J Environment"](#)
- [Section 15.2, "Using the Adobe SVG Viewer to View Performance Charts"](#)

15.1 Introduction to Monitoring the OC4J Environment

The Oracle Enterprise Manager 10g Application Server Control console is a JMX-compliant, web-based user interface that allows you to monitor applications within OC4J. Monitoring involves understanding your OC4J environment, monitoring the OC4J instances in that environment, and monitoring the applications you have deployed. Monitoring OC4J involves understanding your OC4J environment, monitoring the OC4J instances in that environment, and monitoring the applications you have deployed.

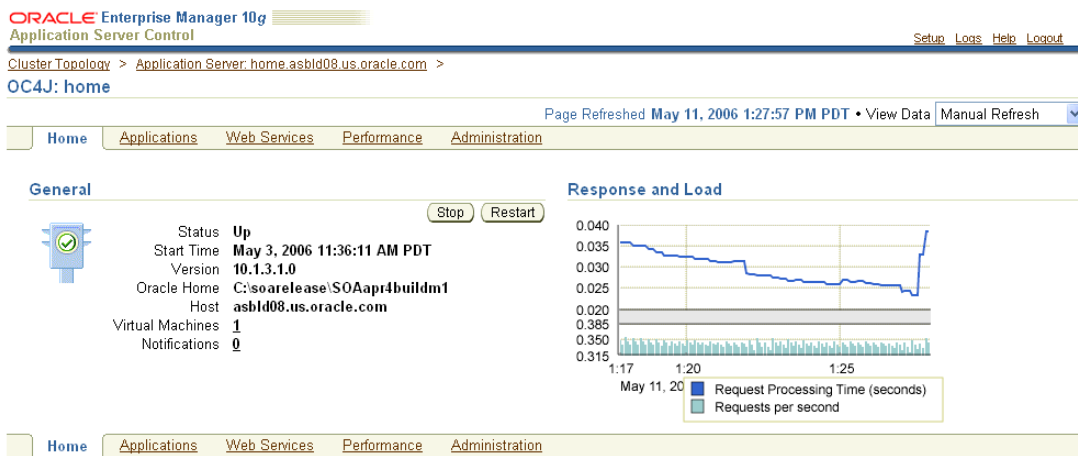
Read this chapter to understand:

- How to Monitor an OC4J Instance
- How to Monitor the Cluster Topology
- How to Monitor Deployed Applications
- How to Monitor Web Module and EJB Module Performance
- How to Monitor JVM Metrics

15.1.1 How to Monitor an OC4J Instance

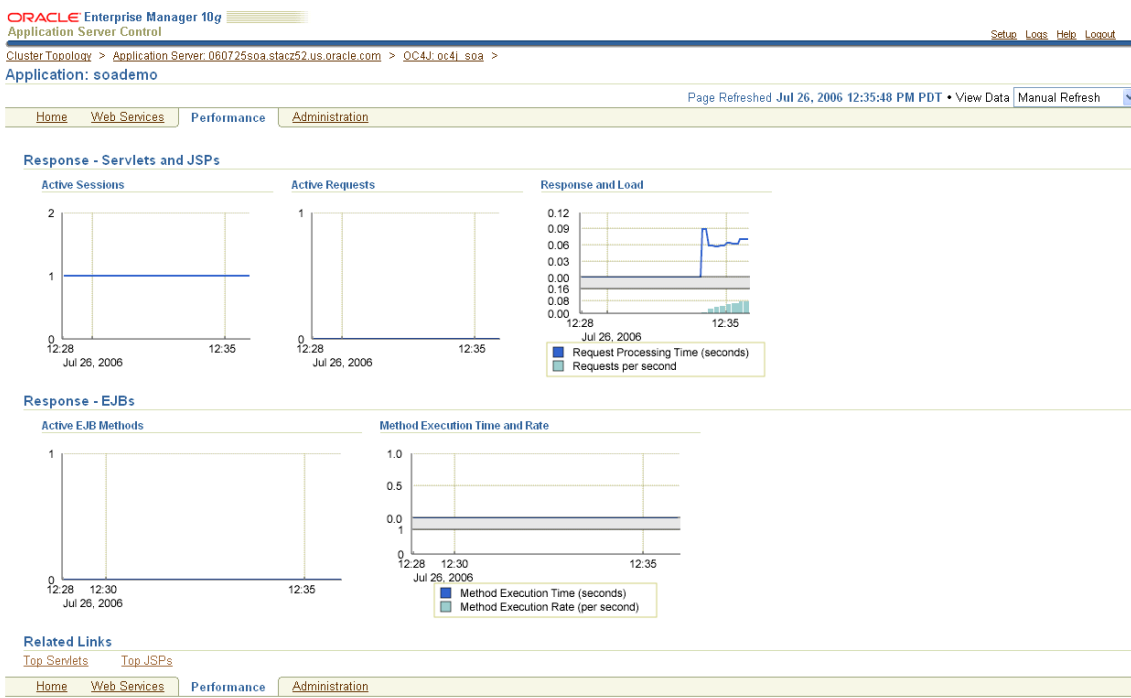
To monitor the performance of a particular OC4J instance, navigate to the OC4J Home page as shown in [Figure 15–1](#).

Figure 15–1 OC4J Home



1. Review the Response and Load chart on the OC4J Home page to obtain a glimpse of how many users are accessing the instance and how quickly the instance is responding to requests. Select a refresh rate from the **View Data** drop-down menu to refresh the Response and Load chart automatically over a period of time.
2. Click **Performance** on the OC4J Home page. Figure 15–2 displays key performance metrics, such as heap usage, Oracle Enterprise Manager JDBC usage, Servlet and JSP performance, and EJB performance within the OC4J instance.

Figure 15–2 OC4J Home: Performance



- From the OC4J Performance page, select a refresh rate from the **View Data** drop-down menu to refresh the data and analyze the value of the performance metrics over a period of time

For more information about the metrics displayed on this page, see Summary of the OC4J Performance Metrics.

15.1.2 How to Monitor the Cluster Topology

Monitoring begins from the Cluster Topology page as shown in Figure 15–3.

Figure 15–3 Cluster Topology

ORACLE Enterprise Manager 10g Application Server Control Setup Logs Help Logout

Cluster Topology Page Refreshed Jul 26, 2006 4:15:39 PM PDT • View Data Manual Refresh

Overview Hosts **1** Application Servers **1**
OC4J Instances **1** HTTP Server Instances **0**

Members View By Applications

[Expand All](#) | [Collapse All](#)

Name	OC4J Status	Instance	Application Server	Host	Context Roots	Routing Enabled	Clustering Enabled	JGroup Protocol	Replication
Applications									
ascontrol									
ccore									
coreman									
CustomerService									
datatags									
default									
esb-dt									
esb-rt									
gateway									
hw_services									
javasso									
orabpel									
orainfra									
policymanager									
ruleauthor									
rulehelp									
SOADEMO	↑	home	060725basic.stac252.us.oracle.com	stac252	/SOADEMO.rmiTunnel	✓			
SOADEMO-CREDITSERVICE-CreditService-WS									
SOADEMO-RAPIDSERVICE-RapidService-WS									
WSIL-App									

↑ Indicates the active ASControl instance.
 TIP Only applications on running OC4Js are shown.

Groups
 A group is a collection of OC4J instances. Certain common management tasks can be performed simultaneously on all OC4J instances in a group. For more information, see [About Groups](#)

[Start](#) [Stop](#) [Delete](#) | [Create](#)

Select Name	OC4J Instance	Status	Application Server
default_group	home	↑	060725basic.stac252.us.oracle.com

Administration

- Cluster MBean Browser
- Routing ID Configuration
- Java SSO Configuration
- Topology Network Configuration
- Runtime Ports

This page provides you with a high-level overview of your OC4J environment. You can use the Cluster Topology page to see how the Oracle Application Server contains components, such as OC4J and Oracle HTTP Server, and how OC4J contains deployed applications.

You can then use the Cluster Topology page to drill down to a specific OC4J instance home page, or to a specific application Home page.

15.1.3 How to Monitor Deployed Applications

If you are interested in a particular application, click the application name on the Cluster Topology page to drill down directly to the application home page.

You can view a list of the applications deployed to a particular OC4J instance by clicking **Applications** on the OC4J Home page. The Applications page also provides a set of key performance metrics for each application that you can scan for a glimpse of how each application is performing. Click the application name to display the Application Home page.

Click **Performance** on the Application Home page to display a series of performance charts. Select a refresh rate from the View Data drop-down menu to refresh the charts automatically over a period of time. Look for spikes and trends in the performance data.

For more information about the metrics displayed on this page, see Summary of the OC4J Performance Metrics.

15.1.4 How to Monitor Web Modules and EJB Module Performance

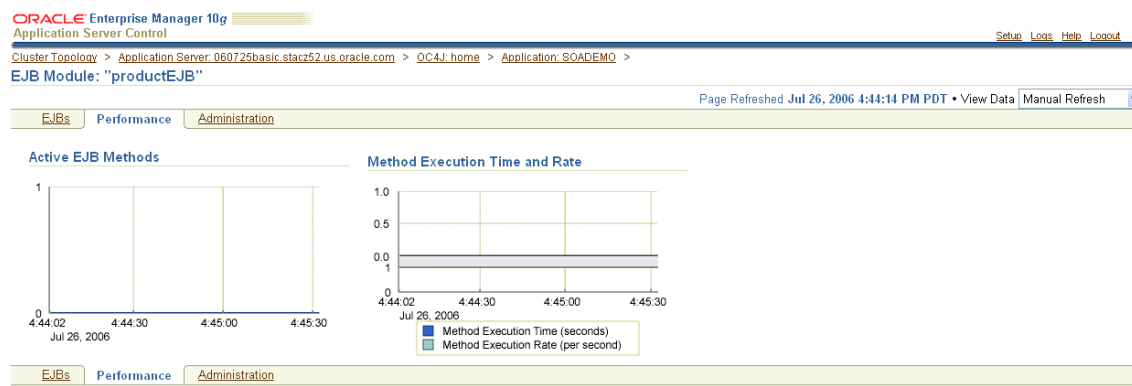
In addition to monitoring deployed applications, you can also monitor individual web modules.

To monitor web and EJB modules:

1. Navigate to the OC4J Home page.
2. Click **Applications** to display the Applications page.
3. Select **Modules** from the View drop-down list.
4. Click the name of the web module or EJB module you want to monitor.
5. Click **Performance**.

Oracle Enterprise Manager displays the Performance page as shown in [Figure 15–4](#). This page provides key performance metrics for the selected Web module or EJB module. For more information about the metrics displayed on this page, see Summary of the OC4J Performance Metrics.

Figure 15–4 EJB Module



15.1.5 How to Monitor JVM Metrics

OC4J executes on the Java Virtual Machine (JVM) of the standard Java Development Kit (JDK). From the Application Server Control Console, you can configure an OC4J instance to run on multiple JVMs, and you can monitor and compare the performance of JVMs associated with the OC4J instance

Oracle application Server offers several methods for monitoring the JVM:

- If you have installed OC4J as part of an Oracle Application Server environment, you can monitor JVM performance using the JVM Metrics page and the OC4J JVM performance page.
- In the standalone OC4J environment, and in the Oracle Application Server environment, you can monitor a JDK 5.0 JVM by setting an OC4J Server property.

15.1.5.1 Monitoring JVM Metrics in the Oracle Application Server Environment

If you are running OC4J in an Oracle Application Server environment, then you can monitor a set of Dynamic Monitoring Service (DMS) metrics for each JVM. These metrics are unavailable in the standalone OC4J environment.

To view the DMS JVM Metrics in the Oracle Application Server environment:

1. Navigate to the OC4J Home page.
2. Locate the Virtual Machines field in the General section of the OC4J Home page.
3. Click the number that indicates how many JVMs are configured for the OC4J instance.

Enterprise Manager displays the JVM Metrics page, which provides a summary of key metrics for all the JVMs configured for the selected OC4J instance. You can use this table to compare the performance of multiple JVMs.

4. For more detailed information, click the name of the JVM you want to monitor.

Enterprise Manager displays the OC4J JVM page, which provides a set of charts and numeric metrics that give you a detailed picture of how the JVM is performing. Use the View Data drop-down list to select a refresh interval. You can then view the changes in the performance charts over a period of time.

15.1.5.2 Monitoring J2SE 5.0 JVM Metrics

Before you can monitor the J2SE 5.0 JVM metrics from the Application Server Control Console, you must be sure that your system meets the following requirements:

- You must be running OC4J on JDK 5.0.
The current OC4J release can run on JDK releases 1.4.2 and 5.0. To monitor the JVM metrics, you must be running on JDK 5.0 (1.5).
- You must make sure that the `jmxremote` system property has been set.

If you are sure your system meets these requirements, you can view the JVM metrics as follows in both the Oracle Application Server environment and in the standalone OC4J environment.

To view the DMS JVM Metrics in the Oracle Application Server environment:

1. Navigate to the OC4J Home page.
2. Locate the **Virtual Machines** field in the General section of the OC4J Home page.
3. Click the number that indicates how many JVMs are configured for the OC4J instance.

Enterprise Manager displays the JVM Metrics page.

4. For more detailed information, click the name of the JVM you want to monitor.
Enterprise Manager displays the OC4J JVM page.
5. Scroll to the Related Links section of the page and click **J2SE 5.0 Metrics**.

To view the DMS JVM Metrics in the standalone OC4J environment:

1. Navigate to the OC4J Home page.
2. Click **Performance** to display the OC4J Performance page.
3. Scroll down to the Related Links section of the page and click **J2SE 5.0 Metrics**.

15.2 Using the Adobe SVG Viewer to View Performance Charts

To help you identify performance issues and trends, Oracle Enterprise Manager Application Server Control provides a variety of performance charts for most of the targets you manage.

Many of the charts are designed to take advantage of the Scalable Vector Graphics (SVG) format, which is an emerging standard supported by the cooperative efforts of the World Wide Web Consortium (W3C) and its members.

As a result, if you download and install the Adobe SVG Viewer, you can experience additional features of the Oracle Enterprise Manager charts. For example, for some charts, the SVG Viewer allows you to right-click and use the context menu to zoom in to a specific area of a chart. In other charts, you can drag and select an area of the chart for closer analysis. For more information on performance tuning, and using multiple components for optimal performance, refer to the *Oracle Application Server Performance Guide*.

For more information, see the SVG Viewer download page on the Adobe Systems Web site.

Note: The SVG plugin is not be available for all Web browsers. Check the Adobe Web site for more information about specific SVG support for your browser. If you do not have the SVG plugin, you can still view Enterprise Manager charts, but you will not be able to take advantage of the additional SVG features.

For more information about the SVG format, see the SVG Information available on the W3C Consortium Web site.

Information about using SVG with Mozilla-based browsers is available in the Mozilla SVG Project Frequently Asked Questions.

Index

A

- action attribute
 - command components, 9-18
 - Property Inspector, 9-22
 - versus `actionListener` attribute, 9-34
- action bindings, 9-6, 9-23
- actions
 - business rules, 8-3
 - pattern matches, 8-23
- activities
 - assign, 7-18
 - decide, 7-36
 - invoke, 7-13
 - receive, 7-15
 - scope, 7-13
 - throw, 7-51
- Activities tab
 - Oracle BPEL Control, 13-9
- activity interactions
 - viewing in Oracle BPEL Control, 13-8
- adapter error handling for Oracle ESB Control, 12-16
- adapter services
 - adding in Oracle Enterprise Service Bus, 6-13
 - configuring in Oracle BPEL Process Manager, 7-28
 - deleting in Oracle Enterprise Service Bus, 6-15
 - modifying in Oracle Enterprise Service Bus, 6-15
- adapters
 - database, 7-28
- ADDRESS table, 2-4
- ADF. *See* Oracle Application Development Framework
- agents in Oracle Web Services Manager, 10-2, 10-4
- application clustering, 11-4
- application scope
 - activities, 3-22
 - defined, 3-2
 - design patterns, using with classification, 3-25
 - designing, 3-21
 - physical design, 3-26
 - process flows, 3-21
 - service interfaces, 3-26
 - services, 3-24
 - standards, 3-27
 - transformations, 3-23

- application server connections, creating for the SOA Order Booking application, 2-9
- application templates for JSF applications, 9-8
- applications
 - creating BPEL, 7-4
 - creating for the web, 9-8
- Assembly project, 2-24
- assign activities
 - Oracle BPEL Process Manager, used in, 7-11, 7-18
 - SOA Order Booking application, used in, 2-34
- asynchronous invocations
 - Oracle BPEL Process Manager, used in, 7-11
 - routing rules, 6-27
 - SOA Order Booking application, used in, 2-39, 2-43
- audit trail, viewing in Oracle BPEL Control, 13-8
- authentication with Oracle Web Services Manager, 10-6
- automatic component binding, 9-14

B

- backing beans
 - about, 9-12
 - automatic component binding, 9-14
 - creating, 9-13
 - introduced, 9-4
 - registering in `faces-config.xml`, 9-13
 - scopes, 9-13
 - web services, invoking, 9-30
- binding container, 9-6
- bindings, Oracle ADF, 9-6
- BPEL processes
 - activities, designing for, 3-22
 - compiling, 7-78
 - creating, 7-4
 - defining, 3-17
 - deploying, 7-78
 - designing, 3-21
 - versions, 7-78
- BPEL Processes tab, Oracle BPEL Control, 13-4
- BPEL projects, 7-4
- BPEL *See* Oracle BPEL Process Manager, business process execution language, 7-1
- branching conditions in BPEL processes, 7-33

- browsing for deployed services in Oracle Enterprise Service Bus, 6-8
- business interaction diagrams, 3-12
- business process execution language, 7-1
- business rules
 - defined, 8-2
 - Oracle BPEL Process Manager, used with decision services, 7-36
 - related terminology, 8-3
 - SOA Order Booking application, used in, 2-37

C

- CallbackClient scope, 2-45
- clearing WSDL cache, Oracle BPEL Control, 13-5
- ClientModel project, 2-24
- compiling
 - BPEL processes, 7-78
 - EJB projects in JDeveloper, 4-25
- conditional branching
 - BPEL processes, 7-33
 - SOA Order Booking applications, used in, 2-36
- connecting, to rules repository, 8-9
- connection factories
 - creating, 11-5
 - for the SOA Order Booking application, 2-7
- connection pools
 - creating, 11-4
 - for the SOA Order Booking application, 2-6
- CreditService project
 - deploying, 2-14
 - directories in, 2-19
- CreditService scope, 2-36
- CRUD methods, 4-7
- CUSTOMER_ADDRESS table, 2-4
- CustomerService (web) project, 2-25
- CustomerService project
 - deploying, 2-14
 - directories in, 2-20
 - tables for, 2-4
- CustomerService scope, 2-35
- CustomerService web service
 - login process, 2-30
 - registration, 2-28

D

- Dashboard tab, Oracle BPEL Control, 13-2
- data binding, 9-5
- data control files, 4-18
- data controls
 - about, 4-18
 - benefits of, 9-5
 - bindings, and, 9-5
 - creating, 4-18
 - from web services, 5-22
- data mapper tool in Oracle Enterprise Service Bus, 6-2

- data model
 - and business rules, 8-3
 - and variables, 8-16
 - and XML schema, 8-11
 - project for in a web application, 9-8
 - web application, 9-8
- data sources
 - creating, 11-4
 - for the SOA Order Booking application, 2-6
- database adapter services
 - adding, 6-13
 - BPEL, in SOA Order Booking application, 2-34
 - ESB, in SOA Order Booking application, 2-43
- database adapters
 - ESB, in SOA Order Booking application, 2-46
 - Oracle BPEL Process Manager, used with, 7-28
- database connections for SOA Order Booking application, 2-9
- databases, for use with SOA Order Booking application, 2-2
- DataControls.dcx file, 4-18, 4-19
- debugging
 - web applications, 9-35
 - web services, 5-24
- decide activities
 - Oracle BPEL Process Manager, used in, 7-36
 - rules dictionary, 8-3
- decision services
 - Oracle BPEL Process Manager, used in, 7-37
 - rules dictionary and, 8-3
 - SOA Order Booking application, used in, 2-37
 - using, 8-24
- default revisions, Oracle BPEL Control, 13-6
- defining trackable fields in Oracle ESB Control, 12-11, 12-12
- deployed services, browsing for in Oracle Enterprise Service Bus, 6-8
- deploying
 - application archive, 11-7
 - applications for the SOA Order Booking application, 2-10
 - BPEL processes, 7-78, 13-5
 - cluster topology, 11-12
 - connection factories, creating, 11-5
 - notifications for JMX events, 11-13
 - Oracle Application Server Control, using, 11-1
 - post deployment, 11-13
 - preparing for, 11-2
 - redeploying applications, 11-13
 - SOA applications, 11-1
 - troubleshooting for applications, 11-15
 - undeploying applications, 11-14
 - versions, 7-78
 - web services, 5-15
- deployment descriptor file, 13-7
- design patterns, 3-25

- designing
 - activities, 3-22
 - application scope, 3-21
 - business interaction diagrams, 3-12
 - design patterns, using with classification, 3-25
 - discover phase, 3-12
 - fault handling, 3-11
 - logical design, 3-21
 - methodologies, introduced, 3-1
 - objectives, determining, 3-7
 - physical design, 3-26
 - process flows
 - defining, 3-17
 - designing, 3-21
 - project definition, 3-15
 - project scope, 3-3
 - requirements, determining, 3-8
 - scopes, 3-2
 - service portfolio, 3-4
 - services
 - analyzing, 3-18
 - classification, 3-19
 - defining, 3-13
 - designing, 3-24
 - discovering, 3-12
 - interfaces, designing, 3-26
 - solution specifications, 3-20
 - standards, 3-27
 - transformations, 3-23
 - use cases, 3-9
- dictionary
 - creating, 8-7
 - definition, 8-3
 - Dictionary Directory field, 8-10
 - Dictionary Name field, 8-10
 - saving, 8-11
- discovery design phase, 3-12
- document transformations, routing rules and, 6-2
- domain decomposition, 3-4
- domains, Oracle BPEL Control, 13-9
- domain-value map in Oracle Enterprise Service Bus, 6-32

E

- EJB (Enterprise JavaBeans)
 - monitoring, 15-4
- EJB_TAB_ID_GEN table, 2-4
- email accounts, configuring, 7-72
- email notifications, 7-67
- EmptyFileRepository, 8-7
- enabling and disabling
 - message tracking in Oracle ESB Control, 12-10
 - trackable fields in Oracle ESB Control, 12-13
- endpoint properties in Oracle Enterprise Service Bus, 6-9
- enterprise scope, 3-2
- entity definition file, 4-18, 4-19

- error handling
 - in Oracle ESB Control, 12-15
 - See also* fault handling
- ESB *See* Oracle Enterprise Service Bus
- execution priority, routing rules and, 6-2
- expression language
 - in JSF pages, 9-4
 - in page definition files, 9-6

F

- faces-config.xml file
 - backing beans, 9-12
 - introduced, 9-4
 - navigation
 - about, 9-5
 - creating, 9-18
 - registering managed beans, 9-13
- facts, business rules and, 8-3
- fault handling
 - designing for, 3-11
 - Oracle BPEL Process Manager, 7-48
 - SOA Order Booking application, 2-45
 - See also* error handling
- FEDEXSHIPMENT table, 2-4
- file repository, 8-9
- files in an Oracle Enterprise Service Bus message flow, 6-5
- filter expressions
 - defined, 6-2
 - routing rules and, 6-2
- ForceType directive, WebDAV content and, 8-8
- FulfillmentESB project
 - deploying, 2-12
 - directories in, 2-21
 - tables for, 2-4

G

- gateways in Oracle Web Services Manager, 10-2, 10-4

H

- header variables
 - viewing in Oracle BPEL Control, 13-9
- HTTP Analyzer in JDeveloper, 5-26
- human workflow tasks
 - input and output parameters, 7-63
 - Oracle BPEL Process Manager, used in, 7-58
 - SOA Order Booking application, used in, 2-39

I

- identity lookup, 7-69
- initiating processes
 - Oracle BPEL Control, 13-3, 13-7
- InsertOrderIntoDB scope, 2-34
- Instances tab
 - Oracle BPEL Control, 13-8

- Instances view in Oracle ESB Control, 12-7
- integration server connections, creating for the SOA Order Booking application, 2-10
- invoke activities, 7-13
- ITEMS table, 2-4

J

- Java Architecture for XML Binding (JAXB)
 - specification, 8-11
- JavaFact, Rule Author and, 8-12
- JavaServer Faces. *See* JSF
- JAXB-generated classes, 8-11
- JMS adapters
 - adding, 6-13
 - SOA Order Booking application, used in, 2-46
- JMSQueue sensor publish type, 7-55
- JMSTopic sensor publish type, 7-55
- JSF (JavaServer Faces)
 - architecture, with Oracle ADF, 9-6
 - automatic component binding, 9-14
 - backing beans
 - about, 9-12
 - creating, 9-13
 - introduced, 9-4
 - libraries for, 9-11
 - managed beans, 9-4
 - navigation diagram
 - creating a JSF page, 9-9
 - creating navigation, 9-17
 - working with, 9-12
 - Oracle ADF, benefits with, 9-6
 - page navigation
 - creating, 9-17
 - introduced, 9-5
 - scopes for managed beans, 9-13
 - templates, files created when using, 9-9
 - workspace, creating, 9-8
- JSF Navigation Modeler, 9-19
- JSF pages
 - automatic component binding, 9-15
 - automatic component binding code, 9-34
 - backing beans for, 9-12
 - creating, 9-9
 - expression language, 9-4
 - web service data controls, using, 9-28
 - web services, collecting data for, 9-28
- JSP (JavaServer Pages)
 - creating, 9-9
 - task display forms, 7-65
- JSP documents, 9-9
- JSR-181
 - with EJBs, 4-16
 - with web services, 5-10

L

- life cycles
 - BPEL processes, 13-4
 - Oracle ADF, 9-25

- load balancing, 11-2
- local interface, 4-9
- locating Web Service WSDL, 10-25
- logging in
 - Oracle BPEL Admin Console, 13-10
 - Oracle BPEL Control, 13-2
 - SOA Order Booking application, 2-28
 - Worklist Application, 7-66
- logical design, 3-21

M

- managed beans
 - application logic in, 9-4
 - automatic component binding and, 9-15
 - backing beans, 9-4
 - configuring in `faces-config.xml` file, 9-13
 - definition of, 9-4
 - introduced, 9-4
 - scopes, 9-13
 - web services, invoking, 9-30
- managing message tracking in Oracle ESB Control, 12-10
- manual recovery, Oracle BPEL Control, 13-5
- Maps view in Oracle ESB Control, 12-7
- message flow
 - creating in Oracle Enterprise Service Bus, 6-3
 - Oracle Enterprise Service Bus files, 6-5
 - processing in Oracle Enterprise Service Bus, 6-4
- `mod_oradav` module, 8-8
- `moddav.conf` file, 8-8
- modifying routing services in Oracle Enterprise Service Bus, 6-23
- monitoring
 - Adobe SVG Viewer, using, 15-6
 - cluster topologies, 15-3
 - deployed applications, 15-4
 - EJB performance, 15-4
 - JVM metrics, 15-5
 - message instances across Oracle Enterprise Service Bus, 12-1
 - OC4J instances, 15-1
 - Oracle BPEL Control, 13-1
 - web modules, 15-4
 - with sensors, 7-54

N

- navigation diagram
 - creating navigation, 9-17
 - using, 9-9
 - working with, 9-12
- navigation. *See* page navigation
- notifications
 - email, 7-67
 - JMX event, 11-13
 - SOA Order Booking application, used in, 2-44
- `NotifyCustomer` scope, 2-44

O

OC4J instance system properties, 8-16

Oracle ADF Faces

introduced, 9-2

libraries, 9-11

Oracle Application Development Framework

architecture, with JSF, 9-6

binding container, 9-6

binding types, 9-6

business technologies supported, 9-3

data binding, 9-5

data controls, 9-2

files created when using, 9-23

forms

creating, 9-26

types, 9-26

input form, creating for web service, 9-28

introduced, 9-1

life cycle, 9-25

page definition file, 9-6

parameter form, 9-28, 9-29

runtime, 9-25

tables

creating, 9-16

types, 9-20

value binding, 9-3

view technologies supported, 9-2

web applications, building with, 9-7

Oracle BPEL Admin Console

logging in, 13-10

Oracle BPEL Control

Activities tab, 13-9

BPEL Processes tab, 13-4

clearing WSDL cache, 13-5

creating reports, 13-7

Dashboard tab, 13-2

default revisions, 13-6

deploying new processes, 13-5

deployment descriptor file, 13-7

domains, 13-9

initiating processes, 13-3, 13-7

Instances tab, 13-8

introduced, 13-1

logging in, 13-2

performing manual recovery, 13-5

refreshing the alarm table, 13-5

test suites, 13-7

undeploying processes, 13-7

viewing activity interactions, 13-8

viewing audit trails, 13-8

viewing header variable details, 13-9

viewing process activities history, 13-8

viewing process logs, 13-5

viewing sensor values, 13-8

Oracle BPEL Control, introduced, 7-80

Oracle BPEL Process Manager, introduced, 7-1

Oracle Enterprise Manager Application Server Control

about, 11-1

application clustering

database replication, 11-4

multicast replication, 11-4

peer-to-peer replication, 11-4

deploying

about, 11-1

application archive, 11-2, 11-7

application clustering, 11-3

ascontrol, 11-7

client module, 11-12

cluster topology, 11-8, 11-12

connection factory, 11-5

connection pools, 11-3, 11-4

connector module, 11-12

context root, 11-11

data sources, 11-3, 11-4

default, 11-10

default, 11-7

deployment plan, 11-10

Deployment Wizard, 11-6

.ear, 11-2, 11-9

EJB JAR file, 11-9

EJB module, 11-12

groups, 11-2

JDBC Resources, 11-4

multiple JVMs, 11-3

opmnctl, 11-6

.rar, 11-5, 11-9

soademo.ear, 11-9

standalone resource adapters, 11-5

.war, 11-2, 11-9

web application, 11-10

load balancing, 11-2

monitoring, 15-1

Adobe SVG Viewer, 15-6

cluster topology, 15-3

deployed applications, 15-4

DMS, 15-5

EJB modules, 15-4

EJB performance, 15-2

heap usage, 15-2

J2SE 5.0 JVM metrics, 15-5

JDK, 15-5

JSP performance, 15-2

JVM metrics, 15-5

OC4J instance, 15-1

Response and Load Chart, 15-2

servlet performance, 15-2

standalone OC4J, 15-5

web modules, 15-4

Oracle Universal Installer, 11-6

port numbers

configuring, 2-16

determining, 2-8

- post-deployment
 - notifications, 11-13
 - redploy, 11-13
 - undeploy, 11-14
- SOA Order Booking application, configuring, 2-5
- Oracle Enterprise Service Bus
 - adding
 - adapter services, 6-13
 - endpoint properties, 6-9
 - SOAP services, 6-11
 - adding routing services, 6-18
 - browsing for deployed services, 6-8
 - configuring ports for the SOA Order Booking application, 2-16
 - creating
 - message flows, 6-3
 - projects, 6-6
 - system or service groups, 6-6
 - deleting
 - adapter services, 6-15
 - routing services, 6-24
 - SOAP services, 6-12
 - features, 6-2
 - files in a message flow, 6-5
 - introduced, 6-1
 - modifying
 - adapter services, 6-15
 - routing services, 6-23
 - SOAP services, 6-12
 - monitoring message instances, 12-1
 - processing in a message flow, 6-4
 - registering projects with an ESB server, 6-38
 - service definitions in Oracle ESB Control, 12-4
 - specifying routing rules, 6-25
 - system definitions in Oracle ESB Control, 12-3
 - testing message flow in Oracle Enterprise Manager
 - Application Server Control, 6-39
 - viewing log files in Oracle Enterprise Manager
 - Application Server Control, 12-14
- Oracle Enterprise Service Bus server
 - registering Oracle Enterprise Service Bus projects and services, 6-38
- Oracle ESB Control
 - adapter error handling, 12-16
 - components, 12-2
 - creating a domain-value map, 6-32
 - defining trackable fields, 12-11, 12-12
 - description of, 12-2
 - enabling and disabling message tracking, 12-10
 - enabling and disabling trackable fields, 12-13
 - error handling in Instances view, 12-15
 - Instances view, 12-7
 - introduced, 12-1
 - managing message tracking, 12-10
 - Maps view, 12-7
 - modifying
 - service definitions, 12-6
 - system definitions, 12-3
 - opening, 12-2

- Oracle Enterprise Service Bus service
 - definitions, 12-4
- Oracle Enterprise Service Bus system
 - definitions, 12-3
- ports, configuring for the SOA Order Booking application, 2-16
- purging message instances, 12-11
- resubmitting messages, 12-16
- searching for message instances, 12-9, 12-10
- Services view, 12-3, 12-4
- viewing all messages in Instances view, 12-8
- Oracle JDeveloper, URL for downloading, 2-3
- Oracle Service Registry, 5-4
- Oracle SOA Suite
 - introduced, 1-3
- Oracle Web Services Manager
 - agents, 10-2
 - authentication use case, 10-6
 - encryption use case, 10-20
 - features, 10-2
 - gateways, 10-2
 - Monitor, 10-3
 - policies, 10-4
 - policy set, 10-8
 - server agents, 10-7
 - use cases, 10-5
- Oracle WSM. *See* Oracle Web Services Manager
- orchestration, designing for, 3-15
- OrderBookingESB flow, 2-32
- OrderBookingESB project
 - deploying, 2-15
 - directories in, 2-21
- OrderBookingFault scope, 2-45
- OrderFulfillmentESB flow, 2-46
- ORDERS table, 2-4
- OrderService project, 2-26

P

- page definition files
 - creating, 9-23
 - expression language in, 9-6
 - introduced, 9-6
- page navigation
 - creating, 9-17
 - details page, creating for, 9-22
 - introduced, 9-5
 - using the JSF Navigation Modeler, 9-19
- parallel flows
 - Oracle BPEL Process Manager, used in, 7-43
 - SOA Order Booking application, used in, 2-43
- partner links
 - Oracle BPEL Process Manager, used in, 7-10
 - SOA Order Booking application, used in, 2-35
- patterns, rules engine and, 8-20
- physical design, 3-26
- policies, in Oracle Web Services Manager, 10-4, 10-8
- policy enforcement points, 10-2
- popup blocking, disabling, 8-20

- port numbers, determining with Application Server Control, 2-8
- PostFulfillmentReq scope, 2-43
- process history, in Oracle BPEL Control, 13-8
- process logs, in Oracle BPEL Control, 13-5
- processing in an Oracle Enterprise Service Bus message flow, 6-4
- PRODUCT table, 2-4, 2-30
- project definition, 3-15
- project scope
 - business interaction diagrams, 3-12
 - defined, 3-2
 - discover phase, 3-12
 - objectives, determining, 3-7
 - process flow definition, 3-17
 - project definition, 3-15
 - requirements, 3-8
 - service portfolio, 3-4
 - services
 - analyzing, 3-18
 - classifying, 3-19
 - defining, 3-13
 - discovering, 3-12
 - solution specification, 3-20
 - summary, 3-3
 - use cases, 3-9
- projects
 - creating for web application in JDeveloper, 9-8
 - creating in Oracle BPEL Process Manager, 7-6
 - creating in Oracle Enterprise Service Bus, 6-6
- proxy server, importing XML schema and, 8-16
- publish types, 7-55
- purging message instances in Oracle ESB Control, 12-11

R

- RapidService project
 - deploying, 2-14
 - directories in, 2-22
- readme.txt file, Oracle Home installation, 8-7
- ReadOnlyCollection.xml file, 4-18
- ReadOnlySingleValue.xml file, 4-18
- receive activities, 7-15
- refreshing the alarm table, in Oracle BPEL Control, 13-5
- registering Oracle Enterprise Service Bus projects and services, 6-38
- remote interface, 4-9
- reports
 - WSI test, 5-26
- reports, in Oracle BPEL Control, 13-7
- repository
 - connecting, 8-9
 - emptyFileRepository, 8-7
 - types supported by Rule Author, 8-7
- RequiresApproval switch, 2-39
- RequiresManualApproval scope, 2-37
- resubmitting messages in Instances view of Oracle ESB Control, 12-16

- RLFact, Rule Author and, 8-12
- routing rules
 - asynchronous or synchronous execution, 6-27
 - defined, 6-2
 - document transformations and, 6-2
 - execution priority and, 6-2
 - expressions for filtering, 6-30
 - filter expressions and, 6-2
 - priority, 6-27
 - specifying in Oracle Enterprise Service Bus, 6-25
 - systems from which messages are accepted, 6-27
 - target operations, 6-26
 - transformations, 6-28
- routing services
 - adding in Oracle Enterprise Service Bus, 6-18
 - defined, 6-2
 - deleting in Oracle Enterprise Service Bus, 6-24
 - modifying in Oracle Enterprise Service Bus, 6-23
- Rule Author
 - assigning role to user, 8-6
 - creating user, 8-5
 - Login page, 8-7
 - starting, 8-6
- rule sets, business rules and, 8-3

S

- scope activities, 7-13
- scopes, for design, 3-2
- searching for message instances in Oracle ESB Control, 12-9, 12-10
- security
 - application-level, 10-1
 - monitoring, 14-1
- SelectManufacturer project
 - deploying, 2-13
 - directories in, 2-23
- SelectSupplier scope, 2-41
- sensors
 - monitoring with, 7-54
 - viewing values in Oracle BPEL Control, 13-8
- service groups
 - creating in Oracle Enterprise Service Bus, 6-6
- Service Oriented Business Applications. *See* SOBAs
- service portfolio plan
 - creating, 3-4
 - defined, 3-3
- service WSDL, locating, 10-25
- service-oriented architecture (SOA), introduced, 1-2
- services
 - building with TopLink, 4-1
 - classifying, 3-19
 - classifying, with design patterns, 3-25
 - defining, 3-18
 - defining in design phase, 3-13
 - designing, 3-24
 - designing interfaces for, 3-26
 - discovering, 3-12
 - as EJBs, 4-1
 - SOA-specific considerations, 3-18

- Services view
 - in Oracle ESB Control, 12-3, 12-4
- session beans
 - creating, 4-7
 - interface types, 4-9
 - updating, 4-11
- session facades
 - about, 4-7
 - generating methods, 4-7
 - updating, 4-11
- SetFinalOrderStatus scope, 2-43
- Simple Object Access Protocol, 5-2
- SOA Order Booking application
 - ADDRESS table, 2-4
 - application server connections for, 2-9
 - assign activities in, 2-34
 - asynchronous invocations in, 2-39, 2-43
 - business rules in, 2-37
 - CallbackClient scope, 2-45
 - conditional branching, 2-36
 - configuring the container for, 2-5
 - connection pools for, 2-6
 - creating data sources for, 2-6
 - CreditService scope, 2-36
 - CUSTOMER_ADDRESS table, 2-4
 - CustomerService scope, 2-35
 - database adapters in
 - BP EL, 2-34, 2-43
 - ESB, 2-46
 - database connection factory for, 2-7
 - database connections for, 2-9
 - database schema
 - about, 2-3
 - installing, 2-3
 - databases for, 2-2
 - decision services in, 2-37
 - deploying, 2-10
 - detailed, 2-28
 - development environment, 2-19
 - downloading, 2-3
 - EJB_TAB_ID_GEN table, 2-4
 - ESB ports, configuring, 2-16
 - fault handling, 2-45
 - FEDEXSHIPMENT table, 2-4
 - human workflows, 2-39
 - InsertOrderIntoDB scope, 2-34
 - installing, 2-2
 - installing the schema, 2-3
 - integration server connections for, 2-10
 - introduced, 2-1
 - ITEMS table, 2-4
 - JMS adapters, 2-46
 - login process, 2-30
 - notifications, 2-44
 - NotifyCustomer scope, 2-44
 - OrderBookingESB flow, 2-32
 - OrderBookingFault scope, 2-45
 - OrderFulfillmentESB flow, 2-46
 - ordering process, 2-30
 - ORDERS table, 2-4
 - overview, 2-17
 - parallel flows, 2-43
 - partner links, 2-35
 - port numbers, determining, 2-8
 - PostFulfillmentReq scope, 2-43
 - predefined users for, 2-28
 - prerequisites, 2-2
 - PRODUCT table, 2-4, 2-30
 - registration process, 2-28
 - RequiresApproval switch, 2-39
 - RequiresManualApproval scope, 2-37
 - running, 2-28
 - securing with Oracle Web Services Manager, 10-5
 - SelectSupplier scope, 2-41
 - SetFinalOrderStatus scope, 2-43
 - SOADEMO-CILENT application, 2-30
 - SOAOrderBooking flow, 2-33
 - switch activities, 2-41
 - synchronous invocations in, 2-35
 - system requirements, 2-2
 - transforming data, 2-34
 - URL for launching, 2-28
- SOADemo application projects, 2-19
- SOADEMO-CLIENT application
 - deploying, 2-15
 - ordering, 2-30
 - tables for, 2-4
 - working with, 2-24
- SOAOrderBooking flow, 2-33
- SOAOrderBooking project
 - deploying, 2-14
 - directories in, 2-23
 - tables for, 2-4
- SOAP
 - see* Simple Object Access Protocol
- SOAP services
 - adding in Oracle Enterprise Service Bus, 6-11
 - deleting in Oracle Enterprise Service Bus, 6-12
 - modifying in Oracle Enterprise Service Bus, 6-12
- SOBAs
 - business interaction diagrams for, 3-12
 - defined, 3-9
 - orchestrating in design, 3-15
- solution specifications, 3-20
- specifying routing rules in Oracle Enterprise Service Bus, 6-25
- SSL, WebDAV and, 8-8
- structure definition file, 4-18, 4-19
- switch activities
 - Oracle BPEL Process Manager, used in, 7-33
 - SOA Order Booking application, used in, 2-41
- synchronous invocations
 - Oracle BPEL Process Manager, used in, 7-11
 - routing rules, 6-27
 - SOA Order Booking application, used in, 2-35
- system properties, OC4J instance, 8-16
- systems in Oracle Enterprise Service Bus, 6-6

T

- target services
 - defined, 6-2
- task display forms, 7-65
- test suites
 - Oracle BPEL Control, 13-7
 - Oracle BPEL Process Manager, 7-79
- testing
 - Oracle BPEL Control, 13-7
 - Oracle BPEL Process Manager, using test suites, 7-79
 - Oracle ESB message flow in Oracle Enterprise Manager Application Server Control, 6-39
 - web applications, 9-35
- throw activities, 7-51
- TopLink, building services with, 4-1
- transformations
 - data mapper in Oracle Enterprise Service Bus, 6-2
 - designing, 3-23
 - Oracle BPEL Process Manager, used in, 7-22
 - reuse across Oracle Enterprise Service Bus, 6-2
 - SOA Order Booking application, used in, 2-34

U

- UDDI
 - see* Universal Description, Discovery, and Integration
- UDDI browser in JDeveloper, 5-4
- undeploying
 - applications, 11-14
- undeploying processes, in Oracle BPEL Control, 13-7
- Universal Description, Discovery, and Integration, 5-4
- UpdateableCollection.xml file, 4-18
- use cases
 - designing with, 3-9
 - for Oracle Web Services Manager, 10-5
- UserInterface project, 2-26
- username tokens in Oracle BPEL Process Manager, 10-20
- users
 - Rule Author, 8-5
 - SOA Order Booking application, 2-28

V

- value binding, Oracle ADF introduced, 9-3
- variables
 - copying, 7-20
 - using in data model, 8-16
- versions
 - of BPEL processes, 7-78
 - overwriting, 7-78, 7-79
 - viewing in Oracle BPEL Control, 13-4

W

- web applications

- Oracle ADF, building with, 9-7
- page navigation, creating, 9-17
- projects created for, 9-8
- testing, 9-35
- workspace, creating, 9-8
- web pages. *See* JSF pages
- web services
 - about, 5-1
 - and JSR-181, 5-10
 - asynchronous, 7-10
 - creating bottom-up, 5-7
 - data controls, using on a page, 9-28
 - debugging locally, 5-24
 - debugging remotely, 5-24
 - deploying, 5-15
 - deployment profile, 5-9
 - development lifecycle, 5-23
 - from EJB 3.0, 5-11
 - interoperability, 5-24
 - invoking
 - from backing beans, 9-30
 - using data controls, 9-28
 - invoking from Oracle BPEL Process Manager, 7-1, 7-10
 - JAX-RPC and Oracle SOAP, 5-8
 - JDeveloper development tools, 5-6
 - proxies, 5-17
 - serialization, 5-6
 - synchronous, 7-10
 - using in applications, 5-15
- Web Services Agent, 10-15
- Web Services Description Language, 5-3
- web services description language, 7-1
- WebDAV repository
 - configuring, 8-8
 - securing, 8-8
- web.xml file, 9-23
- workflow *See* human workflow tasks, 7-58
- Worklist Application, logging in, 7-66
- workspaces, creating for web applications, 9-8
- WSDL
 - see* Web Services Description Language
- WSDL *See* web services description language
- WS-I, 5-24
- WS-I Basic Security Profile, 10-11

X

- XML schema, proxy server and, 8-16
- XMLFact
 - adding, 8-11
 - importing schema, 8-14
- JAXB class directory, 8-12
- JAXB-generated classes, 8-11
- Rule Author and, 8-12
- target package name field, 8-13
- XML Schema field, 8-12
- XPath Expression Builder, 7-67, 7-71
- XPath expressions, 7-19, 7-33
- XSLT Mapper, 7-22

