

Oracle® Fusion Middleware
Developer's Guide for Oracle SOA Suite
11g Release 1 (11.1.1)
E10224-01

May 2009

Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite, 11g Release 1 (11.1.1)

E10224-01

Copyright © 2005, 2009, Oracle and/or its affiliates. All rights reserved.

Primary Author: Virginia Beecher, Deanna Bradshaw, Tulika Das, Mark Kennedy, Alex Prazma, and Peter Purich

Contributor: Oracle SOA Suite development, product management, and quality assurance teams

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

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

If this software or related documentation is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. 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, the use, duplication, disclosure, modification, and adaptation shall be subject to the restrictions and license terms set forth in the applicable Government contract, and, to the extent applicable by the terms of the Government contract, the additional rights set forth in FAR 52.227-19, Commercial Computer Software License (December 2007). Oracle USA, Inc., 500 Oracle Parkway, Redwood City, CA 94065.

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

Oracle is a registered trademark of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

This software and documentation may provide access to or information on content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services.

Contents

Preface	xli
Audience	xli
Documentation Accessibility	xli
Related Documents	xlii
Conventions	xlii
Part I Introduction to Oracle SOA Suite	
1 Introduction to SOA Composite Applications	
1.1 Introduction to Oracle SOA Suite	1-1
1.2 Introduction to SOA Composite Applications	1-3
1.3 Introduction to SCA Technologies	1-4
1.3.1 Binding Components	1-6
1.3.2 Service Infrastructure	1-7
1.3.3 Service Engines and Service Components	1-7
1.3.4 Deployed Service Archives	1-8
1.3.5 Wires	1-8
1.4 Learning Oracle SOA Suite	1-8
2 Overview of SOA Component Editors	
2.1 Introduction to the SOA Composite Editor	2-1
2.1.1 Application Navigator	2-2
2.1.2 Designer	2-3
2.1.3 Left Swim Lane (Exposed Services)	2-3
2.1.4 Right Swim Lane (External References)	2-3
2.1.5 Component Palette	2-3
2.1.6 Resource Palette	2-3
2.1.7 Log Window	2-4
2.1.8 Property Inspector	2-4
2.2 Introduction to the Oracle BPEL Designer	2-4
2.2.1 Application Navigator	2-5
2.2.2 Design Window	2-6
2.2.3 Source Window	2-7
2.2.4 History Window	2-8
2.2.5 Component Palette	2-9

2.2.6	Property Inspector	2-10
2.2.7	Structure Window	2-10
2.2.8	Log Window	2-11
2.3	Introduction to the Oracle Mediator Editor	2-11
2.3.1	Application Navigator	2-12
2.3.2	Mediator Editor	2-13
2.3.3	Source View	2-13
2.3.4	History Window	2-13
2.3.5	Property Inspector	2-14
2.3.6	Structure Window	2-14
2.3.7	Log Window	2-14
2.4	Introduction to the Human Task Editor	2-14
2.4.1	Task Title	2-15
2.4.2	Parameters	2-15
2.4.3	Assignment and Routing Policy	2-15
2.4.4	Expiration and Escalation Policy	2-16
2.4.5	Notification Settings	2-16
2.4.6	Advanced Settings	2-17
2.4.7	Annotations	2-17
2.5	Introduction to the Business Rules Designer	2-18
2.5.1	Application Navigator	2-18
2.5.2	Rules Designer Window	2-18
2.5.3	Structure Window	2-19
2.5.4	Business Rule Validation Log Window	2-20
2.6	Introduction to Oracle Enterprise Manager	2-20

3 Introduction to the SOA Sample Application

3.1	Introduction to the WebLogic Fusion Order Demo Application.....	3-1
3.1.1	The Store Front Module	3-1
3.1.2	The WebLogic Fusion Order Demo Module	3-2
3.2	Setting Up the WebLogic Fusion Order Demo Application.....	3-2
3.3	Taking a Look at the WebLogic Fusion Order Demo Application.....	3-3
3.3.1	Project Applications of the WebLogic Fusion Order Demo Application	3-3
3.3.2	The composite.xml File	3-4

4 Introduction to the Functionality of the SOA Composite Editor

4.1	Introduction to the SOA Composite Editor	4-1
4.2	Designing an SOA Composite Application in Oracle JDeveloper.....	4-1
4.2.1	How to Create an Application and a Project	4-2
4.2.2	How to Add a Service Component	4-5
4.2.3	What You May Need to Know About Adding and Deleting a Service Component	4-7
4.2.4	How to Edit a Service Component.....	4-8
4.2.5	How to Add a Service	4-9
4.2.6	What You May Need to Know About Adding and Deleting Services	4-14
4.2.7	What You May Need to Know About WSDL References.....	4-14
4.2.8	What You May Need to Know About Invoking the Default Revision of a Composite	4-15

4.2.9	How to Wire a Service and a Service Component	4-15
4.2.10	What You May Need to Know About Adding and Deleting Wires	4-16
4.2.11	How to Add a Reference.....	4-17
4.2.12	What You May Need to Know About Adding and Deleting References.....	4-19
4.2.13	How to Wire a Service Component and a Reference	4-19
4.2.14	How to Update Message Schemas of Components (Optional).....	4-21
4.2.15	What You May Need to Know About Updating Message Schemas of Components.....	4-22
4.2.16	How to Invoke Other Composites	4-23
4.2.17	How to Deploy the SOA Composite Application.....	4-24
4.2.18	How to Manage Deployed Composites	4-24
4.2.19	How to Test the SOA Composite Application	4-27

Part II Using the BPEL Process Service Component

5 Getting Started with Oracle BPEL Process Manager

5.1	Introduction to the BPEL Process Service Component	5-1
5.1.1	How to Add a BPEL Process Service Component	5-1
5.2	Introduction to Activities.....	5-4
5.3	Introduction to Partner Links.....	5-5
5.4	Creating a Partner Link	5-6
5.4.1	How to Create a Partner Link	5-7
5.4.1.1	Partner Links for an Outbound Adapter	5-7
5.4.1.2	Partner Links for an Inbound Adapter.....	5-8
5.4.1.3	Partner Links from an Abstract WSDL to Call a Service	5-8
5.4.1.4	Partner Links from an Abstract WSDL to Implement a Service.....	5-8
5.4.1.5	Partner Links and Human Tasks or Business Rules.....	5-9
5.4.1.6	Partner Links from an Existing Human Task, Business Rule, or Oracle Mediator.....	5-9
5.5	Introduction to Technology Adapters	5-10

6 Introduction to Interaction Patterns in a BPEL Process

6.1	Introduction to One-Way Messages.....	6-1
6.2	Introduction to Synchronous Interactions.....	6-2
6.3	Introduction to Asynchronous Interactions.....	6-3
6.4	Introduction to Asynchronous Interactions with a Timeout.....	6-4
6.5	Introduction to Asynchronous Interactions with a Notification Timer.....	6-5
6.6	Introduction to One Request, Multiple Responses	6-6
6.7	Introduction to One Request, One of Two Possible Responses	6-7
6.8	Introduction to One Request, a Mandatory Response, and an Optional Response.....	6-8
6.9	Introduction to Partial Processing	6-9
6.10	Introduction to Multiple Application Interactions	6-10

7 Manipulating XML Data in a BPEL Process

7.1	Introduction to Manipulating XML Data in BPEL Processes.....	7-2
7.1.1	XML Data in BPEL.....	7-2

7.1.2	Data Manipulation and XPath Standards	7-2
7.2	Delegating XML Data Operations to Data Provider Services	7-4
7.2.1	How to Create an Entity Variable	7-6
7.2.1.1	Understanding How SDO Works in the Inbound Direction.....	7-7
7.2.1.2	Understanding How SDO Works in the Outbound Direction	7-7
7.2.1.3	Creating an Entity Variable and Choosing a Partner Link.....	7-8
7.2.1.4	Creating a Binding Key.....	7-9
7.3	Using Standalone SDO-based Variables.....	7-11
7.3.1	How to Declare SDO-based Variables	7-11
7.3.2	How to Convert from XML to SDO	7-12
7.4	Initializing a Variable with Expression Constants or Literal XML.....	7-13
7.4.1	How To Assign a Literal XML Element	7-13
7.5	Copying Between Variables	7-14
7.5.1	How to Copy Between Variables.....	7-14
7.6	Accessing Fields Within Element-Based and Message Type-Based Variables	7-15
7.6.1	How to Access Fields Within Element-Based and Message Type-Based Variables	7-15
7.7	Assigning Numeric Values	7-16
7.7.1	How to Assign Numeric Values	7-16
7.8	Using Mathematical Calculations with XPath Standards	7-16
7.8.1	How To Use Mathematical Calculations with XPath Standards	7-16
7.9	Assigning String Literals.....	7-17
7.9.1	How to Assign String Literals.....	7-17
7.10	Concatenating Strings	7-17
7.10.1	How to Concatenate Strings.....	7-17
7.11	Assigning Boolean Values	7-18
7.11.1	How to Assign Boolean Values	7-18
7.12	Assigning a Date or Time	7-18
7.12.1	How to Assign a Date or Time.....	7-18
7.13	Manipulating Attributes	7-19
7.13.1	How to Manipulate Attributes	7-19
7.14	Manipulating XML Data with bpelx Extensions	7-20
7.14.1	How to Use bpelx:append	7-20
7.14.2	How to Use bpelx:insertBefore	7-21
7.14.3	How to Use bpelx:insertAfter	7-22
7.14.4	How to Use bpelx:remove	7-23
7.14.5	How to Use bpelx:rename and XSD Type Casting	7-24
7.14.6	How to Use bpelx:copyList	7-26
7.15	Validating XML Data with bpelx:validate	7-27
7.15.1	How to Validate XML Data with bpelx:validate.....	7-28
7.16	Manipulating XML Data Sequences That Resemble Arrays	7-28
7.16.1	How to Statically Index into an XML Data Sequence That Uses Arrays.....	7-28
7.16.2	How to Determine Sequence Size	7-29
7.16.3	How to Dynamically Index by Applying a Trailing XPath to an Expression.....	7-29
7.16.3.1	Applying a Trailing XPath to the Result of getVariableData	7-29
7.16.3.2	Using the bpelx:append Extension to Append New Items to a Sequence.....	7-30
7.16.3.3	Merging Data Sequences	7-31
7.16.3.4	Generating Functionality Equivalent to an Array of an Empty Element.....	7-31

7.16.4	What You May Need to Know About SOAP-Encoded Arrays	7-32
7.16.5	What You May Need to Know About Using the Array Identifier	7-32
7.17	Converting from a String to an XML Element.....	7-33
7.17.1	How To Convert from a String to an XML Element.....	7-33
7.18	Understanding the Differences Between Document-Style and RPC-Style WSDL Files	7-33
7.18.1	How To Use RPC-Style Files	7-34
7.19	Manipulating SOAP Headers in BPEL	7-34
7.19.1	How to Receive SOAP Headers in BPEL	7-35
7.19.2	How to Send SOAP Headers in BPEL	7-36
7.20	Using MIME/DIME SOAP Attachments	7-36

8 Invoking a Synchronous Web Service from a BPEL Process

8.1	Introduction to Invoking a Synchronous Web Service.....	8-1
8.2	Invoking a Synchronous Web Service	8-2
8.2.1	How to Invoke a Synchronous Web Service.....	8-2
8.2.2	What Happens When You Invoke a Synchronous Web Service	8-3
8.2.2.1	Partner Link in the BPEL Code.....	8-4
8.2.2.2	Partner Link Type and Port Type in the BPEL Code	8-4
8.2.2.3	Invoke Activity for Performing a Request.....	8-5
8.2.2.4	Synchronous Invocation in BPEL Code	8-5
8.3	Calling a One-Way Mediator with a Synchronous BPEL Process.....	8-6

9 Invoking an Asynchronous Web Service from a BPEL Process

9.1	Introduction to Invoking an Asynchronous Web Service.....	9-1
9.2	Invoking an Asynchronous Web Service	9-2
9.2.1	How to Invoke an Asynchronous Web Service.....	9-2
9.2.1.1	Adding a Partner Link for an Asynchronous Service	9-2
9.2.1.2	Adding an Invoke Activity	9-3
9.2.1.3	Adding a Receive Activity	9-4
9.2.1.4	Performing Additional Activities.....	9-5
9.2.2	What Happens When You Invoke an Asynchronous Web Service	9-5
9.2.2.1	portType Section of the WSDL File.....	9-6
9.2.2.2	partnerLinkType Section of the WSDL File.....	9-6
9.2.2.3	Partner Links Section in the BPEL File	9-7
9.2.2.4	Composite Application File	9-7
9.2.2.5	Invoke and Receive Activities.....	9-7
9.2.2.6	createInstance Attribute for Starting a New Instance	9-8
9.2.2.7	Dehydration Points for Maintaining Long-Running Asynchronous Processes..	9-9
9.2.2.8	Multiple Runtime Endpoint Locations.....	9-9
9.3	Using WS-Addressing in an Asynchronous Service.....	9-9
9.3.1	How to Use WS-Addressing in an Asynchronous Service.....	9-11
9.3.1.1	Using TCP Tunneling to See Messages Exchanged Between Programs	9-11
9.4	Using Correlation Sets in an Asynchronous Service	9-13
9.4.1	How to Use Correlation Sets in an Asynchronous Service.....	9-13
9.4.1.1	Step 1: Creating a Project.....	9-13
9.4.1.2	Step 2: Configuring Partner Links and File Adapter Services	9-14

9.4.1.3	Step 3: Creating Three Receive Activities	9-18
9.4.1.4	Step 4: Creating Correlation Sets.....	9-20
9.4.1.5	Step 5: Associating Correlation Sets with Receive Activities.....	9-20
9.4.1.6	Step 6: Creating Property Aliases.....	9-21
9.4.1.7	Step 7: Reviewing WSDL File Content.....	9-23

10 Using Parallel Flow in a BPEL Process

10.1	Introduction to Parallel Flows in BPEL Processes.....	10-1
10.2	Creating a Parallel Flow	10-2
10.2.1	How to Create a Parallel Flow	10-2
10.2.2	What Happens When You Create a Parallel Flow	10-3
10.3	Customizing the Number of Flow Activities with the flowN Activity	10-5
10.3.1	How to Create a flowN Activity.....	10-6
10.3.2	What Happens When You Create a FlowN Activity.....	10-8

11 Using Conditional Branching in a BPEL Process

11.1	Introduction to Conditional Branching	11-1
11.2	Creating a Switch Activity to Define Conditional Branching	11-2
11.2.1	How to Create a Switch Activity	11-2
11.2.2	What Happens When You Create a Switch Activity	11-3
11.3	Creating a While Activity to Define Conditional Branching.....	11-4
11.3.1	How To Create a While Activity	11-4
11.3.2	What Happens When You Create a While Activity	11-5

12 Using Fault Handling in a BPEL Process

12.1	Introduction to a Fault Handler.....	12-1
12.2	Introduction to BPEL Standard Faults.....	12-3
12.3	Introduction to Categories of BPEL Faults.....	12-3
12.3.1	Business Faults	12-3
12.3.2	Runtime Faults	12-3
12.3.2.1	bindingFault	12-4
12.3.2.2	remoteFault.....	12-4
12.3.2.3	replayFault.....	12-4
12.4	Using the Fault Management Framework	12-4
12.4.1	How to Design a Fault Policy	12-5
12.4.1.1	Understanding How Fault Policy Binding Resolution Works.....	12-6
12.4.1.2	Creating a Fault Policy File for Automated Fault Recovery	12-6
12.4.1.3	Associating a Fault Policy with Fault Policy Binding	12-10
12.4.1.4	Additional Fault Policy and Fault Policy Binding File Samples.....	12-11
12.4.1.5	Designing a Fault Policy with Multiple Rejection Handlers.....	12-14
12.4.2	How to Execute a Fault Policy	12-15
12.4.3	How to Use a Java Action Fault Policy.....	12-15
12.4.4	What You May Need to Know About Fault Management Behavior When the Number of Instance Retries is Exceeded	12-19
12.4.5	What You May Need to Know About Binding Level Retry Execution Within Fault Policy Retries	12-20

12.5	Catching BPEL Runtime Faults	12-21
12.5.1	How to Catch BPEL Runtime Faults.....	12-21
12.6	Getting Fault Details with the getFaultAsString XPath Extension Function	12-21
12.6.1	How to Get Fault Details with the getFaultAsString XPath Extension Function..	12-21
12.7	Throwing Internal Faults	12-22
12.7.1	How to Create a Throw Activity	12-22
12.7.2	What Happens When You Create a Throw Activity	12-23
12.8	Returning External Faults	12-23
12.8.1	How to Return a Fault in a Synchronous Interaction.....	12-23
12.8.2	How to Return a Fault in an Asynchronous Interaction.....	12-24
12.9	Using a Scope Activity to Manage a Group of Activities.....	12-24
12.9.1	How to Create a Scope Activity.....	12-24
12.9.2	What Happens After You Create a Scope Activity.....	12-25
12.9.3	What You May Need to Know About Scopes	12-27
12.9.4	How to Use a Fault Handler within a Scope	12-27
12.9.5	How to Create a Catch Activity	12-28
12.9.6	What Happens When You Create a Catch Branch	12-29
12.9.7	How to Create an Empty Activity to Insert No-Op Instructions into a Business Process	12-30
12.9.8	What Happens When You Create an Empty Activity.....	12-31
12.10	Using Compensation After Undoing a Series of Operations	12-31
12.10.1	How to Use Compensation After Undoing a Series of Operations.....	12-31
12.10.2	How to Create a Compensate Activity	12-32
12.10.3	What Happens When You Create a Compensate Activity.....	12-33
12.11	Using the Terminate Activity to Stop a Business Process Instance	12-33
12.11.1	How to Create a Terminate Activity	12-33
12.11.2	What Happens When You Create a Terminate Activity.....	12-34

13 Incorporating Java and Java EE Code in a BPEL Process

13.1	Introduction to Java and Java EE Code in BPEL Processes	13-1
13.2	Incorporating Java and Java EE Code in BPEL Processes.....	13-1
13.2.1	How to Wrap Java Code as a SOAP Service.....	13-1
13.2.2	What You May Need to Know About Wrapping Java Code as a SOAP Service	13-2
13.2.3	How to Embed Java Code Snippets into a BPEL Process with the bpelx:exec Tag	13-2
13.2.4	How to Use an XML Facade to Simplify DOM Manipulation.....	13-3
13.2.5	How to Use bpelx:exec Built-in Methods.....	13-3
13.2.6	How to Use Java Code Wrapped in a Service Interface.....	13-4
13.3	Adding Custom Classes and JAR Files.....	13-5
13.3.1	How to Add Custom Classes and JAR Files	13-5
13.4	Using Java Embedding in a BPEL Process in Oracle JDeveloper	13-6
13.4.1	How To Use Java Embedding in a BPEL Process in Oracle JDeveloper	13-6
13.5	Embedding Service Data Objects with bpelx:exec	13-7

14 Using Events and Timeouts in BPEL Processes

14.1	Introduction to Event and Timeout Concepts	14-1
14.2	Creating a Pick Activity to Select Between Continuing a Process or Waiting.....	14-1

14.2.1	How To Create a Pick Activity	14-3
14.2.2	What Happens When You Create a Pick Activity	14-4
14.3	Creating a Wait Activity to Set an Expiration Time.....	14-5
14.3.1	How To Create a Wait Activity	14-6
14.3.2	What Happens When You Create a Wait Activity	14-6
14.4	Setting Timeouts for Synchronous Processes	14-6
14.4.1	How To Set Timeouts for Synchronous Processes.....	14-6

15 Coordinating Master and Detail Processes

15.1	Introduction to Master and Detail Process Coordinations	15-1
15.1.1	BPEL File Definition for the Master Process.....	15-3
15.1.1.1	Correlating a Master Process with Multiple Detail Processes	15-5
15.1.2	BPEL File Definition for Detail Processes	15-6
15.2	Defining Master and Detail Process Coordination in Oracle JDeveloper	15-6
15.2.1	How to Create a Master Process.....	15-6
15.2.2	How to Create a Detail Process	15-8
15.2.3	How to Create an Invoke Activity	15-10

16 Using the Notification Service

16.1	Introduction to the Notification Service	16-1
16.2	Introduction to Notification Channel Setup	16-3
16.3	Selecting Notification Channels During BPEL Process Design.....	16-3
16.3.1	How To Configure the Email Notification Channel.....	16-5
16.3.1.1	Setting Email Attachments.....	16-7
16.3.1.2	Formatting the Body of an Email Message as HTML	16-8
16.3.2	How to Configure the IM Notification Channel	16-8
16.3.3	How to Configure the SMS Notification Channel	16-9
16.3.4	How to Configure the Voice Notification Channel	16-11
16.3.5	How to Select Email Addresses and Telephone Numbers Dynamically	16-11
16.3.6	How to Select Notification Recipients by Browsing the User Directory	16-12
16.4	Allowing the End User to Select Notification Channels	16-13
16.4.1	How to Allow the End User to Select Notification Channels	16-13
16.4.1.1	How to Create and Send Headers for Notifications.....	16-14

17 Using Oracle BPEL Process Manager Sensors

17.1	Introduction to Sensors	17-1
17.2	Configuring Sensors and Sensor Actions in Oracle JDeveloper	17-3
17.2.1	How to Configure Sensors	17-3
17.2.2	How to Configure Sensor Actions.....	17-6
17.2.3	How to Publish to Remote Topics and Queues.....	17-9
17.2.4	How to Create a Custom Data Publisher	17-9
17.2.5	How to Register the Sensors and Sensor Actions in composite.xml.....	17-11
17.3	Viewing Sensors and Sensor Action Definitions in Oracle Enterprise Manager Fusion Middleware Control Console.....	17-12

Part III Using the Oracle Mediator Service Component

18 Getting Started with Oracle Mediator

18.1	Introduction to Oracle Mediator.....	18-1
18.2	Overview of Mediator Editor Environment.....	18-3
18.3	Creating a Mediator.....	18-6
18.3.1	Creating a Mediator Without Interface Definition	18-8
18.3.1.1	How to Create a Mediator with No Interface Definition.....	18-8
18.3.1.2	How to Define an Interface for a Mediator with no Interface Definition.....	18-9
18.3.2	Creating a Mediator Based on a WSDL File	18-12
18.3.2.1	How to Create a Mediator Based on a WSDL File.....	18-12
18.3.3	Creating a Mediator with One-Way Interface Definition	18-13
18.3.3.1	How to Create a Mediator with One-Way Interface Definition	18-13
18.3.3.2	What Happens When You Create a Mediator Component with One-Way Interface Definition.....	18-14
18.3.4	Creating a Mediator with Synchronous Interface Definition.....	18-14
18.3.4.1	How to Create a Mediator with Synchronous Interface Definition	18-14
18.3.4.2	What Happens When You Create a Mediator Component with Synchronous Interface Definition	18-15
18.3.5	Creating a Mediator with Asynchronous Interface Definition	18-16
18.3.5.1	How to Create a Mediator with Asynchronous Interface Definition	18-16
18.3.5.2	What Happens When You Create a Mediator Component with Asynchronous Interface Definition.....	18-17
18.3.6	Creating a Mediator Component for Event Subscription	18-17
18.3.6.1	How to Create a Mediator for Event Subscription	18-17
18.3.6.2	What Happens When You Create a Mediator Component for Event Subscription.....	18-20
18.3.7	What You May Need to Know About the Information Available in Mediator User Interface	18-21
18.3.7.1	Mediator Definition.....	18-21
18.3.7.2	Routing Rule.....	18-21
18.4	Generating a WSDL File.....	18-23
18.5	Specifying Operation or Event Subscription Properties	18-25
18.6	Modifying a Mediator Component	18-25
18.6.1	Modifying Operations.....	18-25
18.6.2	Modifying Event Subscriptions	18-27

19 Creating Mediator Routing Rules

19.1	Introduction to Routing Rules	19-1
19.2	Defining Routing Rules.....	19-1
19.2.1	Using the Routing Rules Panel	19-2
19.2.2	Creating Static Routing Rules	19-3
19.2.2.1	Specifying Mediator Services or Events.....	19-4
19.2.2.2	Specifying Sequential or Parallel Execution	19-9
19.2.2.3	Handling Response Messages	19-11
19.2.2.4	Handling Multiple Callbacks.....	19-12
19.2.2.5	Handling Faults	19-12
19.2.2.6	Specifying Expression for Filtering Messages.....	19-15

19.2.2.7	Creating Transformations	19-21
19.2.2.8	Assigning Values	19-22
19.2.2.9	Access Headers for Filters and Assignments	19-25
19.2.2.10	Using Semantic Validation.....	19-27
19.2.2.11	Support for Java Callouts	19-29
19.2.3	Creating Dynamic Routing Rules.....	19-36
19.3	Creating a Mediator for Routing Messages	19-39
19.3.1	Step-By-Step Instructions for Creating the CustomerRouter Use Case.....	19-39
19.3.1.1	Task 1: Creating an Oracle JDeveloper Application and Project.....	19-40
19.3.1.2	Creating CustomerRouter Mediator Component.....	19-40
19.3.1.3	Creating a File Adapter Service.....	19-40
19.3.1.4	Creating a File adapter reference	19-43
19.3.1.5	Specifying Routing Rules	19-44
19.3.1.6	Creating Oracle Application Server Connection	19-51
19.3.1.7	Deploying CustomerRouterProject.....	19-51
19.3.2	Running and Monitoring the CustomerRouterProject Application.....	19-51
19.4	Creating Asynchronous Request Response Using Mediator	19-52
19.4.1	Step-By-Step Instructions for Creating the AsyncMediator Use Case.....	19-52
19.4.1.1	Task 1: Creating an Oracle JDeveloper Application and Project.....	19-53
19.4.1.2	Task 2: Creating a Server BPEL Process	19-53
19.4.1.3	Task 3: Create a Mediator Component.....	19-53
19.4.1.4	Task 4: Creating a Client BPEL Process.....	19-56
19.4.1.5	Task 5: Creating the Invoke, Receive, and Assignment Activities	19-57
19.4.1.6	Task 6: Configuring Oracle Application Server Connection.....	19-61
19.4.1.7	Task 7: Deploying the Composite Application	19-61

20 Using Mediator Error Handling

20.1	Introduction to Oracle Mediator Error Handling	20-1
20.1.1	Fault Policies.....	20-1
20.1.1.1	Conditions	20-2
20.1.1.2	Actions.....	20-4
20.1.2	Fault Bindings	20-6
20.1.3	Error groups in Mediator.....	20-6
20.2	Using Error Handling with Mediator	20-7
20.2.1	How to Use Error Handling for a Mediator Component	20-7
20.2.2	What Happens at Runtime	20-8
20.3	Fault Recovery Using Enterprise Manager	20-8
20.4	Error Handling XML Schema Definition Files	20-8
20.4.1	Schema Definition File for Fault-policies.xml	20-8
20.4.2	Schema Definition File for Fault-bindings.xml	20-12

21 Working with Multiple Part Messages in Mediator

21.1	Introduction to Mediator Multipart Message Support Feature	21-1
21.1.1	Working with Multipart Request Messages	21-2
21.1.1.1	Specifying Filter Expressions.....	21-2
21.1.1.2	Adding Validations	21-2
21.1.1.3	Creating Transformations	21-3

21.1.1.4	Assigning Values	21-4
21.1.2	Working with Multipart Reply, Fault, and Callback Source Messages.....	21-5
21.1.3	Working with Multipart Target Messages.....	21-6

22 Understanding Message Exchange Patterns of a Mediator

22.1	Understanding One-way Message Exchange Pattern	22-2
22.2	Understanding Request-Reply Message Exchange Pattern.....	22-3
22.3	Understanding Request-Reply-Fault Message Exchange Pattern	22-4
22.4	Understanding Request-Callback Message Exchange Pattern.....	22-5
22.5	Understanding Request-Reply-Callback Message Exchange Pattern.....	22-6
22.6	Understanding Request-Reply-Fault-Callback Message Exchange Pattern	22-8

Part IV Using the Business Rules Service Component

23 Using the Business Rule Service Component

23.1	Introduction to the Business Rule Service Component.....	23-1
23.1.1	Integrating BPEL Processes, Business Rules, and Human Tasks	23-2
23.2	Introduction to Creating and Editing Business Rules	23-2
23.2.1	How to Create Business Rules Components	23-2
23.2.2	Introduction to Working with Business Rules in Rules Designer	23-4
23.3	Adding Business Rules to a BPEL Process	23-4
23.3.1	How to Add Business Rules to a BPEL Process	23-4
23.3.2	What Happens When You Add Business Rules to a BPEL Process	23-10
23.3.3	What Happens When You Create a Business Rules Dictionary	23-11
23.3.4	What You Need to Know About Invoking Business Rules in a BPEL Process.....	23-11
23.3.5	What You Need to Know About Decision Component Stateful Operation	23-11
23.4	Adding Business Rules to an SOA Composite Application	23-12
23.4.1	How to Add Business Rules to an SOA Composite Application	23-12
23.4.2	How to Select and Modify a Decision Function in a Business Rule Component..	23-17
23.5	Running Business Rules in a Composite Application	23-19

Part V Using the Human Workflow Service Component

24 Getting Started with Human Workflow

24.1	Introduction to Human Workflow	24-1
24.2	Introduction to Human Workflow Concepts.....	24-3
24.2.1	Introduction to Design and Runtime Concepts	24-3
24.2.1.1	Task Assignment and Routing	24-3
24.2.1.2	Static, Dynamic, and Rule-Based Task Assignment.....	24-6
24.2.1.3	Task Stakeholders.....	24-7
24.2.1.4	Task Deadlines	24-8
24.2.1.5	Notifications	24-8
24.2.1.6	Task Forms	24-9
24.2.1.7	Advanced Concepts	24-9
24.2.1.8	Reports and Audit Trails	24-10

24.2.2	Introduction to the Stages of Human Workflow Design	24-10
24.3	Introduction to Human Workflow Features	24-11
24.3.1	Human Workflow Use Cases	24-11
24.3.1.1	Task Assignment to a User or Role	24-11
24.3.1.2	Use of the Various Participant Types	24-11
24.3.1.3	Escalation, Expiration, and Delegation	24-12
24.3.1.4	Automatic Assignment and Delegation	24-12
24.3.1.5	Dynamic Assignment of Users Based on Task Content.....	24-13
24.3.2	Designing a Human Task from Start to Finish.....	24-13
24.3.2.1	Prerequisites	24-13
24.3.2.2	How to Create the Vacation Request Process.....	24-14
24.3.3	Additional Tutorials	24-27
24.4	Introduction to Human Workflow Architecture	24-27
24.4.1	Human Workflow Services	24-28
24.4.2	Use of Human Task	24-30
24.4.3	Service Engines	24-31

25 Designing Human Tasks

25.1	Introduction to Human Task Design Concepts.....	25-1
25.2	Introduction to the Modeling Process.....	25-1
25.2.1	Create a Human Task Definition.....	25-2
25.2.2	Associate the Human Task Definition with a BPEL Process.....	25-2
25.2.3	Generate the Task Display Form	25-3
25.3	Creating the Human Task Definition with the Human Task Editor.....	25-3
25.3.1	How to Create a Human Task Service Component.....	25-3
25.3.2	What Happens When You Create a Human Task Service Component	25-5
25.3.3	How to Access the Sections of the Human Task Editor.....	25-6
25.3.4	How to Specify the Title, Description, Outcome, Priority, Category, and Owner..	25-7
25.3.4.1	Specifying a Task Title	25-8
25.3.4.2	Specifying a Task Description	25-8
25.3.4.3	Specifying a Task Outcome.....	25-8
25.3.4.4	Specifying a Task Category.....	25-10
25.3.4.5	Specifying a Task Priority	25-10
25.3.4.6	Specifying a Task Owner.....	25-10
25.3.5	How to Specify the Task Payload Data Structure.....	25-16
25.3.6	How to Assign Task Participants	25-18
25.3.6.1	Configuring the Single Participant Type	25-22
25.3.6.2	Configuring the Parallel Participant Type.....	25-30
25.3.6.3	Configuring the Serial Participant Type	25-34
25.3.6.4	Configuring the FYI Participant Type	25-37
25.3.7	How to Select a Routing Policy.....	25-38
25.3.7.1	Routing Tasks to All Participants in the Specified Order.....	25-40
25.3.7.2	Specifying Advanced Task Routing Using Business Rules.....	25-43
25.3.7.3	Using External Routing	25-49
25.3.7.4	Configuring the Error Assignee	25-50
25.3.8	How to Escalate, Renew, or End the Task.....	25-52
25.3.8.1	Introduction to Escalation and Expiration Policy.....	25-52

25.3.8.2	Specifying a Policy to Never Expire	25-53
25.3.8.3	Specifying a Policy to Expire	25-54
25.3.8.4	Extending an Expiration Policy Period	25-54
25.3.8.5	Escalating a Task Policy.....	25-55
25.3.8.6	Specifying a Due Date.....	25-55
25.3.9	How to Specify Participant Notification Preferences	25-56
25.3.9.1	Notifying Recipients of Changes to Task Status	25-57
25.3.9.2	Editing the Notification Message	25-59
25.3.9.3	Setting Up Reminders.....	25-60
25.3.9.4	Changing the Character Set Encoding.....	25-60
25.3.9.5	Securing Notifications to Exclude Details.....	25-60
25.3.9.6	Making Email Messages Actionable.....	25-61
25.3.9.7	Sending Task Attachments with Email Notifications	25-61
25.3.10	How To Specify Advanced Settings.....	25-61
25.3.10.1	Specifying Escalation Rules.....	25-62
25.3.10.2	Specifying WordML Style Sheets for Attachments	25-63
25.3.10.3	Specifying Style Sheets for Attachments.....	25-63
25.3.10.4	Specifying Multilingual Settings	25-63
25.3.10.5	Specifying Callback Classes on Task Status	25-65
25.3.10.6	Specifying a Workflow Signature Policy	25-68
25.3.10.7	Specifying a Certificate Authority.....	25-69
25.3.10.8	Specifying Access Policies on Task Content.....	25-70
25.3.10.9	Specifying Restrictions on Task Assignments.....	25-75
25.3.10.10	Allowing Task and Routing Customization in BPEL Callbacks.....	25-76
25.3.10.11	Showing the Complete Graphical History.....	25-76
25.3.11	How to Specify Annotations	25-76
25.3.12	How to Exit the Human Task Editor and Save Your Changes	25-76
25.4	Associating the Human Task Service Component with a BPEL Process	25-77
25.4.1	How to Associate a Human Task with a BPEL Process	25-77
25.4.2	What You May Need to Know About Deleting a Wire Between a Human Task Service Component and a BPEL Process.....	25-78
25.4.3	How to Define the Human Task Activity Title, Initiator, Priority, and Parameter Variables	25-79
25.4.3.1	Specifying the Task Title.....	25-79
25.4.3.2	Specifying the Task Initiator and Task Priority	25-80
25.4.3.3	Specifying Task Parameters	25-80
25.4.4	How to Define the Human Task Activity Advanced Features	25-82
25.4.4.1	Specifying a Scope Name and a Global Task Variable Name.....	25-83
25.4.4.2	Specifying a Task Owner.....	25-83
25.4.4.3	Specifying an Identification Key	25-83
25.4.4.4	Specifying an Identity Context	25-84
25.4.4.5	Specifying an Application Context	25-84
25.4.4.6	Including the Task History of Other Human Tasks	25-84
25.4.5	How to View the Generated Human Task Activity	25-85
25.4.5.1	Invoking BPEL Callbacks	25-87
25.4.6	What You May Need to Know About Changing the Generated Human Task Activity.....	25-90

25.4.7	What You May Need to Know About Deleting a Partner Link Generated by a Human Task	25-90
25.4.8	How to Define Outcome-Based Modeling.....	25-91
25.4.8.1	Specifying Payload Updates	25-91
25.4.8.2	Using Case Statements for Other Task Conclusions	25-91

26 Designing Task Display Forms for Human Tasks

26.1	Introduction to the Task Display Form	26-1
26.2	Associating the Task Flow with the Task Service	26-2
26.3	Creating an ADF Task Flow Based on a Human Task	26-3
26.3.1	How To Autogenerate an ADF Task Flow for a Human Task	26-3
26.3.2	How To Create an ADF Task Flow Based on a Human Task	26-4
26.3.3	What Happens When You Create an ADF Task Flow Based on a Human Task	26-5
26.4	Creating a Task Display Form	26-6
26.4.1	How To Create a Task Display Form Using the Complete Task with Payload Drop Handler	26-11
26.4.2	How To Create Task Display Form Regions Using Individual Drop Handlers ...	26-13
26.4.3	How To Add the Payload to the Task Display Form	26-15
26.4.4	What Happens When You Create a Task Display Form.....	26-17
26.5	Refreshing Data Controls When the Task XSD Changes	26-17
26.6	Securing the Task Flow Application	26-18
26.7	Creating an Email Notification	26-18
26.7.1	How To Create an Email Notification	26-18
26.7.1.1	Creating a Task Flow with a Router	26-19
26.7.1.2	Creating an Email Notification Page	26-22
26.7.2	What Happens When You Create an Email Notification Page.....	26-25
26.7.3	What You May Need to Know About Creating an Email Notification Page.....	26-25
26.8	Deploying a Composite Application with a Task Flow	26-25
26.8.1	Before Deploying the Task Display Form: Port Changes	26-25
26.8.2	How To Deploy a Composite Application with a Task Flow	26-27
26.8.3	How To Redeploy the Task Display Form.....	26-27
26.8.4	How To Deploy a Task Flow as a Separate Application.....	26-27
26.8.5	How To Deploy a Task Display Form to a non-SOA Oracle WebLogic Server	26-28
26.8.5.1	Deploying oracle.soa.workflow.jar to a non-SOA Oracle WebLogic Server ..	26-28
26.8.5.2	Defining the Foreign JNDI Provider on a non-SOA Oracle WebLogic Server.....	26-30
26.8.5.3	Defining the Foreign JNDI Provider Links on a non-SOA Oracle WebLogic Server.....	26-32
26.8.5.4	Including a Grant for bpm-services.jar.....	26-34
26.8.5.5	Deploying the Application.....	26-35
26.8.6	What Happens When You Deploy the Task Display Form.....	26-35
26.9	Displaying a Task Display Form in the Worklist	26-35
26.9.1	How To Display the Task Display Form in the Worklist	26-36
26.10	Displaying a Task in an Email Notification	26-36

27 Using Oracle BPM Worklist

27.1	Introduction to Oracle BPM Worklist	27-1
------	---	------

27.1.1	What You May Need To Know About Oracle BPM Worklist.....	27-3
27.2	Logging In to Oracle BPM Worklist	27-3
27.2.1	How To Log In to the Worklist.....	27-3
27.2.1.1	Enabling the weblogic User for Logging in to the Worklist.....	27-4
27.2.2	What Happens When You Log In to the Worklist.....	27-4
27.3	Customizing the Task List Page	27-7
27.3.1	How To Filter Tasks	27-7
27.3.2	How To Create and Customize Worklist Views	27-14
27.3.3	How To Customize the Task Status Chart.....	27-18
27.3.4	How To Create a ToDo Task.....	27-19
27.3.5	How To Create a Subtask	27-20
27.4	Acting on Tasks: The Task Details Page.....	27-21
27.4.1	System Actions.....	27-24
27.4.2	Task History	27-24
27.4.3	How To Act on Tasks	27-27
27.4.4	How To Act on Tasks That Require a Digital Signature.....	27-34
27.5	Approving Tasks.....	27-37
27.6	Setting a Vacation Period.....	27-38
27.7	Setting Rules	27-39
27.7.1	How To Create User Rules	27-40
27.7.2	How To Create Group Rules.....	27-41
27.7.3	Assignment Rules for Tasks with Multiple Assignees.....	27-43
27.8	Using the Worklist Administration Functions	27-43
27.8.1	How To Manage Other Users' or Groups' Rules (as an Administrator).....	27-43
27.8.2	How To Set the Worklist Display (Application Preferences).....	27-44
27.9	Specifying Notification Settings.....	27-46
27.9.1	Messaging Filter Rules	27-46
27.9.1.1	Data Types.....	27-46
27.9.1.2	Attributes.....	27-46
27.9.2	Rule Actions.....	27-47
27.9.3	Managing Messaging Channels.....	27-47
27.9.3.1	Viewing Your Messaging Channels.....	27-48
27.9.3.2	Creating, Editing, and Deleting a Messaging Channel.....	27-49
27.9.4	Managing Messaging Filters	27-49
27.9.4.1	Viewing Messaging Filters	27-50
27.9.4.2	Creating Messaging Filters.....	27-50
27.9.4.3	Editing a Messaging Filter.....	27-52
27.9.4.4	Deleting a Messaging Filter.....	27-52
27.10	Using Flex Fields	27-52
27.10.1	How To Map Flex Fields.....	27-53
27.11	Creating Worklist Reports	27-56
27.11.1	How To Create Reports	27-57
27.11.2	What Happens When You Create Reports	27-58
27.11.2.1	Unattended Tasks Report.....	27-59
27.11.2.2	Tasks Priority Report	27-60
27.11.2.3	Tasks Cycle Time Report.....	27-60
27.11.2.4	Tasks Productivity Report.....	27-61

27.12	Accessing Oracle BPM Worklist in Local Languages	27-62
27.12.1	How To Change the Language Used in the Worklist.....	27-62
27.12.2	How To Change the Time Zone Used in the Worklist.....	27-63

28 Building a Custom Worklist Client

28.1	Introduction to Building Clients for Workflow Services	28-1
28.2	Packages and Classes for Building Clients.....	28-2
28.3	Workflow Service Clients	28-3
28.3.1	The IWorkflowServiceClient Interface	28-5
28.4	Class Paths for Clients Using SOAP.....	28-6
28.5	Class Paths for Clients Using Remote EJBs.....	28-6
28.6	Class Paths for Clients Using Local EJBs.....	28-7
28.7	Enterprise JavaBeans References in Web Applications.....	28-7
28.8	Initiating a Task.....	28-7
28.8.1	Creating a Task.....	28-8
28.8.2	Creating a Payload Element in a Task.....	28-8
28.8.3	Initiating a Task Programmatically.....	28-9
28.9	Changing Workflow Standard View Definitions.....	28-10
28.10	Writing a Worklist Application Using the HelpDeskUI Sample	28-10

29 Introduction to Human Workflow Services

29.1	Introduction to Human Workflow Services.....	29-1
29.1.1	Enterprise JavaBeans, SOAP, and Java Support for the Human Workflow Services.....	29-1
29.1.2	Security Model for Services.....	29-3
29.1.2.1	Limitation on Propagating Identity to Workflow Services when Using SOAP Web Services.....	29-4
29.1.2.2	Creating Human Workflow Context on Behalf of a User.....	29-4
29.1.3	Task Service	29-4
29.1.4	Task Query Service.....	29-7
29.1.5	Identity Service.....	29-9
29.1.5.1	Identity Service Providers	29-10
29.1.6	Task Metadata Service	29-11
29.1.7	User Metadata Service.....	29-12
29.1.8	Task Report Service	29-14
29.1.9	Runtime Config Service	29-14
29.1.9.1	Internationalization of Attribute Labels.....	29-16
29.1.10	Evidence Store Service and Digital Signatures.....	29-17
29.1.10.1	Prerequisites	29-19
29.1.10.2	Interfaces and Methods	29-19
29.1.11	Task Instance Attributes	29-21
29.2	Notifications from Human Workflow	29-25
29.2.1	Contents of Notification.....	29-26
29.2.2	Error Message Support	29-27
29.2.3	Reliability Support.....	29-27
29.2.4	Management of Oracle Human Workflow Notification Service	29-28
29.2.5	How to Configure the Notification Channel Preferences.....	29-28

29.2.6	How to Configure Notification Messages in Different Languages	29-29
29.2.7	How to Send Actionable Messages	29-30
29.2.7.1	How to Send Actionable Emails for Human Tasks	29-30
29.2.8	How to Send Inbound and Outbound Attachments	29-31
29.2.9	How to Send Inbound Comments.....	29-32
29.2.10	How to Send Secure Notifications.....	29-32
29.2.11	How to Set Channels Used for Notifications.....	29-32
29.2.12	How to Send Reminders.....	29-32
29.2.13	How to Set Automatic Replies to Unprocessed Messages	29-33
29.2.14	How to Create Custom Notification Headers	29-34
29.3	Assignment Service Configuration	29-34
29.3.1	Dynamic Assignment and Task Escalation Functions	29-34
29.3.1.1	How to Implement a Dynamic Assignment Function	29-35
29.3.1.2	How to Configure Dynamic Assignment Functions.....	29-36
29.3.1.3	How to Configure Display Names for Dynamic Assignment Functions.....	29-37
29.3.1.4	How to Implement a Task Escalation Function	29-37
29.3.2	Dynamically Assigning Task Participants with the Assignment Service	29-37
29.3.2.1	How to Implement an Assignment Service.....	29-38
29.3.2.2	Example of Assignment Service Implementation.....	29-39
29.3.2.3	How to Deploy a Custom Assignment Service.....	29-41
29.3.3	Custom Escalation Function.....	29-41
29.4	Class Loading for Callbacks and Resource Bundles.....	29-41
29.5	Resource Bundles in Workflow Services.....	29-42
29.5.1	Task Resource Bundles	29-42
29.5.2	Global Resource Bundle – WorkflowLabels.properties	29-42
29.5.3	Worklist Client Resource Bundles.....	29-44
29.5.4	Task Detail ADF Task Flow Resource Bundles.....	29-44
29.5.5	Case Sensitivity	29-45
29.6	Introduction to Human Workflow Client Integration with Oracle WebLogic Server Services	29-45
29.6.1	Human Workflow Services Clients.....	29-45
29.6.1.1	Task Query Service Client Code.....	29-46
29.6.1.2	Configuration Option	29-49
29.6.1.3	Client Logging.....	29-51
29.6.1.4	Configuration Migration Utility	29-51
29.6.2	Identity Propagation	29-52
29.6.2.1	Enterprise JavaBeans Identity Propagation.....	29-52
29.6.2.2	SAML Token Identity Propagation for SOAP Client.....	29-52
29.6.3	Client JAR Files	29-54
29.7	Database Views for Oracle Workflow.....	29-54
29.7.1	Unattended Tasks Report View.....	29-54
29.7.2	Task Cycle Time Report View.....	29-55
29.7.3	Task Productivity Report View	29-56
29.7.4	Task Priority Report View	29-56

30 Integrating Microsoft Excel with a Human Task

30.1	Configuring Your Environment for Invoking a BPEL Process from an Excel
------	--

Workbook.....	30-1
30.1.1 How to Create an JDeveloper Project of the Type Web Service Data Control	30-1
30.1.2 How to Create a Dummy JSF Page	30-2
30.1.3 How to Add Desktop Integration to Your Oracle JDeveloper Project.....	30-2
30.1.4 What Happens When You Add Desktop Integration to Your JDeveloper Project.	30-2
30.1.5 How to Deploy the Web Application You Created in Step 1.....	30-4
30.1.6 How to Install Microsoft Excel.....	30-4
30.1.7 How to Install the Oracle Oracle ADF-Desktop Integration Plug-in.....	30-4
30.1.8 How to Specify the User Interface Controls and Create the Excel Workbook	30-4
30.2 Attaching Excel Workbooks to Human Task Workflow Email Notifications	30-4
30.2.1 Enabling Attachment of Excel Workbooks to Human Task Workflow Email Notifications	30-4
30.2.2 What Happens During Runtime When You Enable Attachment of Excel Workbooks to Human Task Workflow Email Notifications	30-5
30.2.3 Example: Attaching an Excel Workbook to Email Notifications	30-5
30.2.3.1 Task 1: Enable the ADF Task Flow Project with Oracle ADF-DI Capabilities .	30-5
30.2.3.2 Task 2: Set up Authentication	30-10
30.2.3.3 Task 3: Create a Valid Page Definition File to Be Used in the Excel Workbook	30-12
30.2.3.4 Task 4: Prepare the Excel Workbook	30-17
30.2.3.5 Task 5: Deploy the ADF Task Flow	30-21
30.2.3.6 Task 6: Test the Deployed Application	30-23

Part VI Using Oracle Business Activity Monitoring

31 Creating Oracle BAM Data Objects

31.1 Introduction to Creating Data Objects.....	31-1
31.2 Defining Data Objects.....	31-2
31.2.1 How to Define a Data Object	31-2
31.2.2 How to Add Columns to a Data Object.....	31-2
31.2.3 How to Add Lookup Columns to a Data Object.....	31-3
31.2.4 How to Add Calculated Columns to a Data Object.....	31-4
31.2.5 How to Add Time Stamp Columns to a Data Object	31-5
31.2.6 What You May Need to Know About System Data Objects	31-5
31.2.7 What You May Need to Know About Oracle Data Integrator Data Objects	31-6
31.3 Creating Permissions on Data Objects.....	31-6
31.3.1 How to Create Permissions on a Data Object.....	31-6
31.3.2 How to Add a Group of Users.....	31-7
31.3.3 How to Copy Permissions from Other Data Objects.....	31-7
31.4 Viewing Existing Data Objects.....	31-7
31.4.1 How to View Data Object General Information.....	31-8
31.4.2 How to View Data Object Layouts.....	31-8
31.4.3 How to View Data Object Contents	31-9
31.5 Using Data Object Folders	31-9
31.5.1 How to Create Folders	31-9
31.5.2 How to Open Folders.....	31-10
31.5.3 How to Set Folder Permissions.....	31-10

31.5.4	How to Move Folders.....	31-11
31.5.5	How to Rename Folders	31-11
31.5.6	How to Delete Folders	31-12
31.6	Creating Security Filters.....	31-12
31.6.1	How to Create a Security Filter.....	31-12
31.6.2	How to Copy Security Filters from Other Data Objects	31-13
31.7	Creating Dimensions	31-14
31.7.1	How to Create a Dimension.....	31-14
31.7.2	How to Create a Time Dimension.....	31-15
31.8	Renaming and Moving Data Objects	31-16
31.8.1	How to Rename a Data Object.....	31-16
31.8.2	How to Move a Data Object.....	31-16
31.9	Creating Indexes	31-16
31.9.1	How to Create an Index.....	31-16
31.10	Clearing Data Objects.....	31-17
31.10.1	How to Clear a Data Object.....	31-17
31.11	Deleting Data Objects.....	31-17
31.11.1	How to Delete a Data Object.....	31-17

32 Integrating Oracle BAM with SOA Composite Applications

32.1	Introduction to Integrating Oracle BAM with SOA Composite Applications.....	32-1
32.2	Configuring Oracle BAM Adapter	32-2
32.3	Creating a Design Time Connection to an Oracle BAM Server	32-2
32.3.1	How to Create a Connection to an Oracle BAM Server	32-2
32.4	Using Oracle BAM Adapter in an SOA Composite Application.....	32-3
32.4.1	How to Use Oracle BAM Adapter in an SOA Composite Application	32-3
32.5	Using Oracle BAM Adapter in a BPEL Process.....	32-4
32.5.1	How to Use Oracle BAM Adapter in a BPEL Process	32-4
32.6	Integrating BPEL Sensors with Oracle BAM	32-6
32.6.1	How to Create a Sensor.....	32-6
32.6.2	How to Create an Oracle BAM Sensor Action	32-7
32.6.3	How to Disable an Oracle BAM Sensor Action.....	32-9

33 Creating Oracle BAM Enterprise Message Sources

33.1	Introduction to Enterprise Message Sources	33-1
33.2	Creating Enterprise Message Sources.....	33-2
33.2.1	How to Create an Enterprise Message Source.....	33-2
33.2.2	How to Configure DateTime Specification.....	33-6
33.2.3	How to Use Advanced XML Formatting	33-8
33.3	Using Foreign JMS Providers.....	33-9
33.4	Use Case: Creating an EMS Against Oracle Streams AQ JMS Provider.....	33-10
33.4.1	Creating a JMS Topic in AQ-JMS	33-10
33.4.2	Creating a Data Source in Oracle WebLogic Server	33-12
33.4.3	Creating a Foreign JMS Server.....	33-12
33.4.4	Defining an EMS in Oracle BAM Architect	33-13
33.4.5	Inserting and Updating Records in the SQL Table.....	33-14

34 Using Oracle Data Integrator With Oracle BAM

34.1	Introduction to Using the Oracle Data Integrator With Oracle Business Activity Monitoring	34-1
34.2	Installing the Oracle Data Integrator Integration Files.....	34-2
34.2.1	How to Install Integration Files Using the Script.....	34-2
34.2.2	How to Manually Install Integration Files	34-4
34.3	Creating the Oracle BAM Target	34-6
34.3.1	How to Create the Oracle BAM Target	34-6
34.4	Using Oracle BAM Knowledge Modules	34-7
34.5	Updating the Oracle Data Integrator External Data Source Definition	34-13
34.5.1	How to Update the Oracle Data Integrator External Data Source Definitions.....	34-13
34.6	Launching Oracle Data Integrator Scenarios From Oracle BAM Alerts.....	34-14

35 Creating External Data Sources

35.1	Introduction to External Data Sources.....	35-1
35.2	Creating External Data Sources	35-1
35.2.1	How to Create an External Data Source.....	35-2
35.2.2	What You May Need to Know About Oracle Data Integrator External Data Sources	35-2
35.2.3	How to Edit an External Data Source	35-2
35.2.4	How to Delete an External Data Source	35-2

36 Using Oracle BAM Web Services

36.1	Introduction to Oracle BAM Web Services	36-1
36.2	Using the DataObjectOperations Web Services	36-2
36.2.1	How to Use the DataObjectOperations Web Services.....	36-2
36.3	Using the DataObjectDefinition Web Service.....	36-3
36.3.1	How to Use the DataObjectDefinition Web Service	36-3
36.4	Using the ManualRuleFire Web Service.....	36-4
36.4.1	How to Use the ManualRuleFire Web Service	36-4
36.5	Using the ICommand Web Service	36-4
36.5.1	How to Use the ICommand Web Service.....	36-5

37 Creating Oracle BAM Alerts

37.1	Introduction to Creating Alerts.....	37-1
37.2	Creating Alert Rules	37-2
37.2.1	How to Create an Alert Rule.....	37-2
37.2.2	How to Activate Alerts	37-3
37.2.3	How to Modify Alert Rules.....	37-4
37.2.4	How to Delete an Alert	37-4
37.3	Creating Alert Rules From Templates	37-4
37.3.1	How to Create Alert Rules From Templates.....	37-4
37.4	Creating Alert Rules With Messages	37-5
37.4.1	How to Create an Alert Rule With a Message.....	37-5
37.5	Creating Complex Alerts	37-6
37.5.1	How to Create a Dependent Rule.....	37-6

37.6	Using Alert History	37-6
37.6.1	How to View Alert History	37-6
37.6.2	How to Clear Alert History	37-6
37.7	Launching Alerts by Invoking Web Services.....	37-7

38 Using ICommand

38.1	Introduction to ICommand	38-1
38.2	Executing ICommand.....	38-1
38.3	Specifying the Command and Option Syntax	38-2
38.3.1	How to Specify the Security Credentials.....	38-2
38.3.2	How to Specify the Command.....	38-3
38.3.3	How to Specify Object Names	38-3
38.3.4	How to Specify Multiple Parameter Targets	38-4
38.4	Using Command-line-only Parameters.....	38-5
38.5	Running ICommand Remotely	38-6

Part VII Using Oracle User Messaging Service

39 Oracle User Messaging Service

39.1	User Messaging Service Overview	39-1
39.1.1	Components.....	39-2
39.1.2	Architecture	39-2

40 Sending and Receiving Messages using the User Messaging Service Java API

40.1	Overview of UMS Java API.....	40-1
40.1.1	Creating a Java EE Application Module.....	40-1
40.2	Creating a UMS Client Instance.....	40-2
40.2.1	Creating a MessagingEJBClient Instance Using a Programmatic or Declarative Approach 40-2	
40.2.2	API Reference for Class MessagingClientFactory.....	40-2
40.3	Sending a Message.....	40-2
40.3.1	Creating a Message.....	40-3
40.3.1.1	Creating a Plaintext Message.....	40-3
40.3.1.2	Creating a Multipart/Alternative Message (with Text/Plain and Text/HTML Parts).....	40-3
40.3.1.3	Creating Delivery Channel-Specific Payloads in a Single Message for Recipients with Different Delivery Types.....	40-3
40.3.2	API Reference for Class MessageFactory	40-4
40.3.3	API Reference for Interface Message	40-4
40.3.4	API Reference for Enum DeliveryType	40-4
40.3.5	Addressing a Message	40-4
40.3.5.1	Types of Addresses	40-5
40.3.5.2	Creating Address Objects.....	40-5
40.3.5.3	Creating a Recipient with a Failover Address.....	40-5
40.3.5.4	API Reference for Class AddressFactory	40-5

40.3.5.5	API Reference for Interface Address	40-5
40.3.6	Retrieving Message Status.....	40-6
40.3.6.1	Synchronous Retrieval of Message Status	40-6
40.3.6.2	Asynchronous Notification of Message Status	40-6
40.4	Receiving a Message.....	40-6
40.4.1	Registering an Access Point	40-6
40.4.2	Synchronous Receiving.....	40-7
40.4.3	Asynchronous Receiving.....	40-7
40.4.4	Message Filtering	40-7
40.5	Using the UMS Enterprise JavaBeans Client API to Build a Client Application.....	40-7
40.5.1	Overview of Development	40-8
40.5.2	Configuring the Email Driver	40-8
40.5.3	Using JDeveloper 11g to Build the Application	40-9
40.5.3.1	Opening the Project.....	40-9
40.5.4	Deploying the Application	40-11
40.5.5	Testing the Application.....	40-12
40.6	Using the UMS Enterprise JavaBeans Client API to Build a Client Echo Application	40-15
40.6.1	Overview of Development	40-15
40.6.2	Configuring the Email Driver	40-16
40.6.3	Using JDeveloper 11g to Build the Application	40-16
40.6.3.1	Opening the Project.....	40-16
40.6.4	Deploying the Application	40-20
40.6.5	Testing the Application.....	40-20
40.7	Creating a New Application Server Connection.....	40-22

41 Parlay X Web Services Multimedia Messaging API

41.1	Overview of Parlay X Messaging Operations.....	41-1
41.2	Send Message Interface.....	41-2
41.2.1	sendMessage Operation.....	41-2
41.2.2	getMessageDeliveryStatus Operation	41-3
41.3	Receive Message Interface	41-3
41.3.1	getReceivedMessages Operation.....	41-4
41.3.2	getMessage Operation.....	41-5
41.3.3	getMessageURIs Operation.....	41-5
41.4	Oracle Extension to Parlay X Messaging.....	41-6
41.4.1	ReceiveMessageManager Interface	41-6
41.4.1.1	startReceiveMessage Operation	41-6
41.4.1.2	stopReceiveMessage Operation.....	41-7
41.5	Parlay X Messaging Client API and Client Proxy Packages.....	41-7
41.6	Sample Chat Application with Parlay X APIs	41-8
41.6.1	Overview	41-8
41.6.1.1	Provided Files	41-9
41.6.2	Running the Pre-Built Sample	41-9
41.6.3	Testing the Sample.....	41-12
41.6.4	Creating a New Application Server Connection.....	41-16

42 User Messaging Preferences

42.1	Introduction	42-1
42.1.1	Terminology	42-1
42.1.2	Configuration of Notification Delivery Preferences	42-2
42.1.3	Delivery Preference Rules	42-2
42.1.3.1	Data Types	42-3
42.1.3.2	System Terms	42-3
42.1.3.3	Business Terms.....	42-3
42.1.4	Rule Actions.....	42-4
42.2	How to Manage Messaging Channels	42-5
42.2.1	Creating a Channel	42-5
42.2.2	Editing a Channel	42-6
42.2.3	Deleting a Channel	42-7
42.2.4	Setting a Default Channel.....	42-7
42.3	Creating Contact Rules using Filters.....	42-7
42.3.1	Creating Filters.....	42-9
42.3.2	Editing a Filter.....	42-11
42.3.3	Deleting a Filter.....	42-11
42.4	Configuring Settings.....	42-11

Part VIII Sharing Functionality Across Oracle SOA Suite Components

43 Deploying SOA Composite Applications

43.1	Creating an Application Server Connection	43-1
43.2	Deploying a Single SOA Composite in Oracle JDeveloper	43-2
43.2.1	How to Deploy a Single SOA Composite	43-2
43.2.1.1	Optionally Creating a Project Deployment Profile	43-2
43.2.1.2	Deploying the Profile	43-3
43.2.2	What You May Need to Know About Oracle JDeveloper Deployment to a Managed Oracle WebLogic Server.....	43-7
43.2.3	What You May Need to Know About Invoking References in One-Way SSL Environments in Oracle JDeveloper.....	43-7
43.3	Deploying Multiple SOA Composite Applications in Oracle JDeveloper	43-8
43.3.1	How to Deploy Multiple SOA Composite Applications	43-8
43.4	Deploying and Using Shared Metadata Across SOA Composite Applications	43-10
43.4.1	How to Deploy Shared Metadata.....	43-11
43.4.1.1	Create a JAR Profile and Include the Artifacts to Share	43-11
43.4.1.2	Create a SOA Bundle that Includes the JAR Profile	43-16
43.4.1.3	Deploy the SOA Bundle	43-18
43.4.2	How to Use Shared Metadata	43-18
43.4.2.1	Create a SOA-MDS Connection	43-18
43.4.2.2	Create a BPEL Process	43-19
43.5	Deploying an Existing SOA Archive in Oracle JDeveloper.....	43-21
43.5.1	How to Deploy an Existing SOA Archive from Oracle JDeveloper.....	43-21
43.6	Managing SOA Composite Applications with Scripts	43-22
43.6.1	How to Manage SOA Composite Applications with the WLST Utility	43-22

43.6.2	How to Manage SOA Composite Applications with ant Scripts.....	43-22
43.6.2.1	Testing a SOA Composite Application	43-23
43.6.2.2	Compiling a SOA Composite Application	43-24
43.6.2.3	Packaging a SOA Composite Application into a Composite SAR file	43-25
43.6.2.4	Deploying SOA Composite Application.....	43-25
43.6.2.5	Undeploying a SOA Composite Application.....	43-26
43.6.2.6	Managing a SOA Composite Application	43-27
43.6.2.7	Upgrading a SOA Composite Application.....	43-29
43.6.2.8	How to Manage SOA Composite Applications with ant Scripts	43-29
43.7	Moving SOA Composite Applications to and from Development, Test, and Production Environments.....	43-30
43.7.1	Introduction to Configuration Plans.....	43-30
43.7.2	Introduction to a Configuration Plan File	43-31
43.7.3	Introduction to Use Cases for a Configuration Plan.....	43-32
43.7.4	How to Create a Configuration Plan in Oracle JDeveloper.....	43-33
43.7.5	How to Create a Configuration Plan with the WLST Utility	43-36

44 Using Business Events and the Event Delivery Network

44.1	Introduction to Business Events	44-1
44.1.1	Local and Remote Events Boundaries	44-3
44.2	Creating Business Events in Oracle JDeveloper	44-3
44.2.1	How to Create a Business Event.....	44-4
44.2.2	How to Subscribe to a Business Event.....	44-6
44.2.3	What Happens When You Create and Subscribe to a Business Event	44-8
44.2.4	What You May Need to Know About Subscribing to a Business Event	44-8
44.2.5	How to Publish a Business Event.....	44-9
44.2.6	What Happens When You Publish a Business Event.....	44-9
44.2.7	How to Integrate Oracle ADF Business Component Business Events with Oracle Mediator	44-10

45 Creating Transformations with the XSLT Mapper

45.1	Introduction to the XSLT Mapper	45-1
45.1.1	Overview of XSLT Creation	45-3
45.1.2	Guidelines for Using the XSLT Mapper	45-6
45.2	Creating an XSL Map File	45-6
45.2.1	How to Create an XSL Map File in Oracle BPEL Process Manager	45-6
45.2.2	How to Create an XSL Map File from Imported Source and Target Schema Files in Oracle BPEL Process Manager	45-8
45.2.3	How to Create an XSL Map File in Oracle Mediator.....	45-10
45.2.4	What You May Need to Know About Creating an XSL Map File.....	45-13
45.2.5	What Happens at Runtime If You Pass a Payload Through Oracle Mediator Without Creating an XSL Map File	45-14
45.3	Designing Transformation Maps with the XSLT Mapper	45-14
45.3.1	How to Add Additional Sources	45-14
45.3.2	How to Perform a Simple Copy by Linking Nodes.....	45-16
45.3.3	How to Set Constant Values.....	45-16
45.3.4	How to Add Functions.....	45-17

45.3.4.1	Editing Function Parameters	45-18
45.3.4.2	Chaining Functions	45-19
45.3.4.3	Using Named Templates	45-19
45.3.4.4	Importing User-Defined Functions.....	45-20
45.3.5	How to Edit XPath Expressions.....	45-22
45.3.6	How to Add XSLT Constructs	45-24
45.3.6.1	Using Conditional Processing with <code>xsl:if</code>	45-25
45.3.6.2	Using Conditional Processing with <code>xsl:choose</code>	45-26
45.3.6.3	Creating Loops with <code>xsl:for-each</code>	45-27
45.3.6.4	Cloning <code>xsl:for-each</code>	45-28
45.3.6.5	Applying <code>xsl:sort</code> to <code>xsl:for-each</code>	45-28
45.3.6.6	Copying Nodes with <code>xsl:copy-of</code>	45-29
45.3.6.7	Including External Templates with <code>xsl:include</code>	45-30
45.3.7	How to Automatically Map Nodes.....	45-30
45.3.7.1	Using Auto Mapping with Confirmation	45-32
45.3.8	What You May Need to Know About Automatic Mapping	45-33
45.3.9	How to View Unmapped Target Nodes	45-34
45.3.10	How to Generate Dictionaries.....	45-35
45.3.11	How to Create Map Parameters and Variables.....	45-36
45.3.11.1	Creating a Map Parameter	45-36
45.3.11.2	Creating a Map Variable.....	45-37
45.3.12	How to Search Source and Target Nodes	45-38
45.3.13	How to Control the Generation of Unmapped Target Elements.....	45-39
45.3.14	How to Ignore Elements in the XSLT Document.....	45-39
45.3.15	How to Replace a Schema in the XSLT Mapper.....	45-39
45.3.16	How to Substitute Elements and Types in the Source and Target Trees.....	45-40
45.4	Testing the Map.....	45-43
45.4.1	How to Test the Transformation Mapping Logic	45-44
45.4.2	How to Generate Reports	45-46
45.4.2.1	Correcting Memory Errors When Generating Reports.....	45-47
45.4.3	How to Customize Sample XML Generation	45-48
45.5	Demonstrating the New Features of the XSLT Mapper.....	45-48
45.5.1	Opening the Application	45-49
45.5.2	Creating a New XSLT Map in the BPEL Process	45-49
45.5.3	Using Type Substitution to Map the Purchase Order Items	45-50
45.5.4	Referencing Additional Source Elements.....	45-51
45.5.5	Using Element Substitution to Map the Shipping Address	45-53
45.5.6	Mapping the Remaining Fields	45-54
45.5.7	Testing the Map	45-55

46 Working with Domain Value Maps

46.1	Introduction to Domain Value Maps.....	46-1
46.1.1	Domain Value Map Features	46-2
46.1.1.1	Qualifier Support.....	46-2
46.1.1.2	Qualifier Order Support	46-3
46.1.1.3	One-to-Many Mapping Support	46-4
46.2	Creating Domain Value Maps.....	46-4

46.2.1	How to Create Domain Value Maps.....	46-4
46.2.2	What Happens When You Create a Domain Value Map	46-5
46.3	Editing a Domain Value Map.....	46-7
46.3.1	Adding Columns to a Domain Value Map	46-7
46.3.2	Adding Rows to a Domain Value Map	46-8
46.4	Using Domain Value Map Functions.....	46-8
46.4.1	Understanding Domain Value Map Functions	46-8
46.4.1.1	dvm:lookupValue.....	46-8
46.4.1.2	dvm:lookupValue1M.....	46-9
46.4.2	Using Domain Value Map Functions in Transformation	46-9
46.4.3	Using Domain Value Map Functions in XPath Expressions	46-11
46.4.4	What Happens at Runtime.....	46-12
46.5	Creating a Domain Value Map Use Case for Hierarchical Lookup	46-12
46.5.1	Creating the HierarchicalValue Use Case	46-13
46.5.1.1	Task 1: Creating an Oracle JDeveloper Application and Project.....	46-13
46.5.1.2	Task 2: Creating a Domain Value Map	46-14
46.5.1.3	Task 3: Creating a File Adapter Service	46-15
46.5.1.4	Task 4: Creating ProcessOrders Mediator Component	46-16
46.5.1.5	Task 5: Creating a File Adapter Reference.....	46-17
46.5.1.6	Task 6: Specifying Routing Rules.....	46-18
46.5.1.7	Task 7: Configuring Oracle Application Server Connection.....	46-21
46.5.1.8	Task 8: Deploying the Composite Application	46-21
46.5.2	Running and Monitoring the HierarchicalValue Application	46-21
46.6	Creating a Domain Value Map Use Case For Multiple Values.....	46-22
46.6.1	Creating the Multivalue Use Case.....	46-22
46.6.1.1	Task 1: Creating an Oracle JDeveloper Application and Project.....	46-22
46.6.1.2	Task 2: Creating a Domain Value Map	46-23
46.6.1.3	Task 3: Creating a File Adapter Service	46-24
46.6.1.4	Task 4: Creating LookupMultiplevaluesMediator Mediator Component.....	46-25
46.6.1.5	Task 5: Creating a File Adapter Reference.....	46-26
46.6.1.6	Task 6: Specifying Routing Rules.....	46-27
46.6.1.7	Task 7: Configuring Oracle Application Server Connection.....	46-30
46.6.1.8	Task 8: Deploying the Composite Application	46-30
46.6.2	Running and Monitoring the Multivalue Application.....	46-30

47 Working with Cross References

47.1	Introduction to Cross References.....	47-1
47.2	Creating and Modifying Cross Reference Tables.....	47-3
47.2.1	Creating a Cross Reference Table.....	47-4
47.2.1.1	What Happens When You Create a Cross Reference.....	47-5
47.2.2	Adding an End System to a Cross Reference Table.....	47-6
47.3	Populating Cross Reference Tables	47-6
47.3.1	About xref:populateXRefRow Function.....	47-7
47.3.2	About xref:populateXRefRow1M Function	47-9
47.3.3	How to Populate a Column of a Cross Reference Table	47-10
47.4	Looking Up Cross Reference Tables	47-12
47.4.1	About xref:lookupXRef Function	47-12

47.4.2	About xref:lookupXRef1M Function	47-13
47.4.3	About xref:lookupPopulatedColumns Function	47-14
47.4.4	How to Look Up a Cross Reference Table for a Value.....	47-14
47.5	Deleting a Cross Reference Table Value	47-16
47.5.1	How to Delete a Cross Reference Table Value	47-17
47.6	Creating and Running Cross Reference Use Case	47-18
47.6.1	Step-By-Step Instructions for Creating the Use Case	47-19
47.6.1.1	Task 1: Configuring Oracle Database and Database Adapter	47-19
47.6.1.2	Task 2: Creating an Oracle JDeveloper Application and Project.....	47-20
47.6.1.3	Task 3: Creating a Cross Reference	47-21
47.6.1.4	Task 4: Creating a Database Adapter Service.....	47-22
47.6.1.5	Task 5: Creating EBS and SBL External References.....	47-24
47.6.1.6	Task 6: Creating Logger External Reference.....	47-26
47.6.1.7	Task 7: Creating Mediator Components	47-28
47.6.1.8	Task 8: Specifying Routing Rules for Mediator Component	47-28
47.6.1.9	Task 9: Specifying Routing Rules for Common Mediator.....	47-38
47.6.1.10	Task 10: Configuring Oracle Application Server Connection.....	47-49
47.6.1.11	Task 11: Deploying the Composite Application	47-49
47.6.2	Running and Monitoring the XrefCustApp Application.....	47-49
47.7	Creating and Running Cross Reference for 1M Functions	47-50
47.7.1	Step-By-Step Instructions for Creating the Use Case	47-50
47.7.1.1	Task 1: Configuring Oracle Database and Database Adapter	47-51
47.7.1.2	Task 2: Creating an Oracle JDeveloper Application and Project.....	47-52
47.7.1.3	Task 3: Creating a Cross Reference	47-52
47.7.1.4	Task 4: Creating a Database Adapter Service.....	47-53
47.7.1.5	Task 5: Creating EBS External Reference	47-55
47.7.1.6	Task 6: Creating Logger External Reference.....	47-57
47.7.1.7	Task 7: Creating Mediator Components	47-58
47.7.1.8	Task 8: Specifying Routing Rules for Mediator Component	47-59
47.7.1.9	Task 9: Specifying Routing Rules for Common Mediator.....	47-63
47.7.1.10	Task 10: Configuring Oracle Application Server Connection.....	47-68
47.7.1.11	Task 11: Deploying the Composite Application	47-68

48 Using Two-Layer Business Process Management (BPM)

48.1	Introduction to Two-Layer Business Process Management	48-1
48.2	Phase Activities	48-3
48.2.1	Creating a Phase Activity	48-3
48.2.2	How to Create a Phase Activity.....	48-3
48.2.3	What Happens When You Create a Phase Activity.....	48-4
48.2.4	What Happens at Runtime When You Create a Phase Activity.....	48-5
48.2.5	What You May Need to Know About Creating a Phase Activity	48-5
48.3	The Dynamic Routing Decision Table	48-5
48.3.1	How to Create the Routing Decision Table	48-6
48.3.2	What Happens When You Create the Routing Decision Table	48-7
48.4	Use Case: Two-Layer BPM	48-7
48.4.1	Designing the SOA Composite	48-7
48.4.2	Creating a Phase Activity	48-9

48.4.3	Creating and Editing the Dynamic Routing Decision Table	48-10
48.4.4	Adding Assign Activities to the BPEL Process Model	48-11
48.4.5	Deploying the Sample with JDeveloper	48-12
48.4.5.1	Creating an Application Deployment Profile.....	48-12
48.4.5.2	Creating an Application Server Connection.....	48-13
48.4.5.3	Deploying the Application.....	48-13

49 Testing SOA Composite Applications

49.1	Introduction to the Composite Test Framework.....	49-1
49.1.1	Test Cases Overview	49-1
49.1.2	Test Suites Overview	49-1
49.1.3	Emulations Overview	49-2
49.1.4	Assertions Overview	49-2
49.2	Introduction to the Components of a Test Suite.....	49-2
49.2.1	Process Initiation.....	49-3
49.2.2	Emulations	49-3
49.2.3	Assertions.....	49-4
49.2.4	Message Files	49-5
49.3	Creating Test Suites and Test Cases	49-5
49.3.1	How to Create Test Suites and Test Cases	49-5
49.4	Creating the Contents of Test Cases.....	49-8
49.4.1	How to Initiate Inbound Messages	49-8
49.4.2	How to Emulate Outbound Messages.....	49-10
49.4.3	How to Emulate Callback Messages.....	49-13
49.4.4	How to Emulate Fault Messages	49-15
49.4.5	How to Create Assertions.....	49-16
49.4.5.1	Creating Assertions on a Part Section, Nonleaf Element, or Entire XML Document	49-17
49.4.5.2	Creating Assertions on a Leaf Element	49-20
49.4.6	What You May Need to Know About Assertions.....	49-22
49.5	Deploying and Running a Test Suite	49-23

50 Managing Policies

50.1	Introduction to Policies	50-1
50.2	Attaching Policies to Binding Components and Service Components	50-2
50.2.1	How to Attach Policies to Binding Components and Service Components	50-2

51 Defining Composite Sensors

51.1	Introduction to Composite Sensors	51-1
51.1.1	Restrictions on Use of Composite Sensors	51-1
51.2	Adding Composite Sensors	51-2
51.2.1	How to Add Composite Sensors	51-2
51.2.2	Adding a Variable.....	51-5
51.2.3	Adding an Expression.....	51-5
51.2.4	Adding a Property	51-6
51.3	Monitoring Composite Sensor Data During Runtime.....	51-6

52 Using Service Data Objects and Enterprise JavaBeans

52.1	Introduction to SDO and Enterprise JavaBeans Binding	52-1
52.2	Designing an Enterprise JavaBeans Application.....	52-2
52.2.1	How to Create SDO Objects Using the SDO Compiler.....	52-2
52.2.2	How to Create a Session Bean and Import the SDO Objects.....	52-3
52.2.3	How to Create a Profile and an EAR File.....	52-3
52.2.4	How to Define the SDO Types with an Enterprise JavaBeans Bean	52-3
52.2.5	How to Use Web Service Annotations	52-5
52.2.6	How to Deploy the Enterprise JavaBeans EAR File	52-6
52.3	Creating an Enterprise JavaBeans Adapter Service in Oracle JDeveloper	52-6
52.3.1	Invoking SDO-based Enterprise JavaBeans from SOA Composite Applications ...	52-6
52.3.1.1	How to Invoke SDO-based Enterprise JavaBeans from SOA Composite Applications	52-6
52.3.2	Invoking SOA Composite Applications from Enterprise JavaBeans using SDO Parameters	52-8
52.3.2.1	How to Invoke SOA Composite Applications from Enterprise JavaBeans using SDO Parameters.....	52-8
52.4	Designing an Enterprise JavaBeans Client to Invoke Oracle SOA Suite	52-9
52.5	Specifying Enterprise JavaBeans Roles	52-10
52.6	Configuring JNDI Access.....	52-10
52.6.1	How to Create a Foreign JNDI.....	52-10
52.6.2	How to Create a Custom CSF Map for JNDI Lookup	52-11

53 Processing Large Documents

53.1	Introduction to Processing Large Documents	53-1
53.2	Best Practices for Handling Large Documents.....	53-1
53.2.1	Setting Audit Levels from Oracle Enterprise Manager for Large Payload Processing	53-2
53.2.2	Using the Assign Activity in BPEL/Mediator.....	53-2
53.2.3	Using XSLT Transformations for Repeating Structures.....	53-2
53.2.4	Using Adapter Support for Streaming Large Payloads	53-2
53.2.5	Using Correct Settings for Large Payload Scenarios	53-3
53.2.6	Processing Large Documents in Oracle B2B.....	53-3
53.2.6.1	MDSInstance Cache Size	53-3
53.2.6.2	Protocol Message Size.....	53-3
53.2.6.3	Number of Threads	53-4
53.2.6.4	StuckThread Max Time.....	53-4
53.2.6.5	Tablespace.....	53-4
53.2.7	Setting the Default JTA Timeout in for Large Documents	53-5
53.2.8	Using Large Number of Activities in BPEL Processes (Without FlowN).....	53-5
53.2.9	Using Large Number of Activities in BPEL Processes (With FlowN)	53-5
53.2.10	Using a Flow With Multiple Sequences	53-5
53.2.11	Using a Flow with One Sequence.....	53-5
53.2.12	Using Flow with No Sequence.....	53-6
53.2.13	Large Numbers of Mediators in Composites	53-6
53.2.14	Using XSLT Transformations on Large Payloads (For BPEL and Mediator)	53-6

53.3	Limitations on Concurrent Processing of Large Documents.....	53-6
53.3.1	Opaque Schema for Processing Large Payloads	53-6
53.3.2	Streaming MTOM Attachments	53-6
53.3.3	Importing Large Data Sets in Oracle B2B.....	53-6

Part IX Appendices

A BPEL Process Activities and Services

A.1	Introduction to Activities and Components	A-1
A.2	Introduction to BPEL Activities	A-2
A.2.1	Tabs Common to Many Activities.....	A-2
A.2.2	Assign Activity.....	A-3
A.2.3	Bind Entity Activity.....	A-4
A.2.4	Compensate Activity.....	A-5
A.2.5	Create Entity	A-6
A.2.6	Email Activity.....	A-6
A.2.7	Empty Activity	A-7
A.2.8	Flow Activity	A-8
A.2.9	FlowN Activity.....	A-9
A.2.10	IM Activity.....	A-10
A.2.11	Invoke Activity.....	A-11
A.2.12	Java Embedding Activity.....	A-12
A.2.13	Phase Activity.....	A-12
A.2.14	Pick Activity	A-13
A.2.15	Receive Activity	A-14
A.2.16	Receive Signal Activity	A-15
A.2.17	Remove Entity Activity.....	A-16
A.2.18	Reply Activity.....	A-16
A.2.19	Scope Activity.....	A-17
A.2.20	Sequence Activity	A-18
A.2.21	Signal Activity	A-19
A.2.22	SMS Activity	A-19
A.2.23	Switch Activity.....	A-20
A.2.24	Terminate Activity.....	A-21
A.2.25	Throw Activity	A-21
A.2.26	Transform Activity	A-22
A.2.27	User Notification.....	A-23
A.2.28	Voice Activity	A-24
A.2.29	Wait Activity	A-24
A.2.30	While Activity	A-25
A.3	Introduction to BPEL Services	A-26
A.3.1	AQ Adapter	A-27
A.3.2	Oracle B2B.....	A-27
A.3.3	Oracle BAM Adapter.....	A-27
A.3.4	Database Adapter	A-27
A.3.5	File Adapter	A-27
A.3.6	FTP Adapter	A-27

A.3.7	JMS Adapter	A-27
A.3.8	MQ Adapter.....	A-28
A.3.9	Oracle Applications	A-28
A.3.10	Partner Link (Web Service/Adapter)	A-28
A.3.11	Socket Adapter	A-29
A.4	Publishing and Browsing the Oracle Service Registry	A-29
A.4.1	How to Publish a Business Service	A-29
A.4.2	How to Add a Binding Template	A-30
A.4.3	How to Create a Connection to the Registry	A-30
A.4.4	How to Configure a SOA project to Invoke a Service from the Registry	A-30
A.4.5	How To Configure the Inquiry URL for Runtime	A-31
A.5	Validating When Loading a Process Diagram.....	A-32

B XPath Extension Functions

B.1	SOA XPath Extension Functions.....	B-1
B.1.1	Database Functions.....	B-1
B.1.1.1	lookup-table.....	B-1
B.1.1.2	query-database.....	B-2
B.1.1.3	sequence-next-val	B-2
B.1.2	Date Functions.....	B-3
B.1.2.1	add-dayTimeDuration-to-dateTime	B-3
B.1.2.2	current-date	B-3
B.1.2.3	current-dateTime	B-4
B.1.2.4	current-time	B-4
B.1.2.5	day-from-dateTime	B-4
B.1.2.6	format-dateTime	B-5
B.1.2.7	hours-from-dateTime.....	B-5
B.1.2.8	implicit-timezone.....	B-5
B.1.2.9	minutes-from-dateTime.....	B-6
B.1.2.10	month-from-dateTime	B-6
B.1.2.11	seconds-from-dateTime	B-6
B.1.2.12	subtract-dayTimeDuration-from-dateTime.....	B-6
B.1.2.13	timezone-from-dateTime.....	B-7
B.1.2.14	year-from-dateTime	B-7
B.1.3	Mathematical Functions.....	B-7
B.1.3.1	abs	B-8
B.1.4	String Functions	B-8
B.1.4.1	compare.....	B-8
B.1.4.2	compare-ignore-case	B-9
B.1.4.3	create-delimited-string.....	B-9
B.1.4.4	ends-with	B-9
B.1.4.5	format-string	B-10
B.1.4.6	get-content-as-string	B-10
B.1.4.7	get-content-from-file-function	B-10
B.1.4.8	get-localized-string.....	B-11
B.1.4.9	index-within-string.....	B-11
B.1.4.10	last-index-within-string	B-12

B.1.4.11	left-trim	B-12
B.1.4.12	lower-case	B-13
B.1.4.13	matches.....	B-13
B.1.4.14	right-trim.....	B-13
B.1.4.15	upper-case.....	B-14
B.2	BPEL XPath Extension Functions	B-14
B.2.1	addQuotes.....	B-14
B.2.2	appendToList.....	B-14
B.2.3	copyList	B-15
B.2.4	countNodes.....	B-16
B.2.5	doc.....	B-16
B.2.6	doStreamingTranslate	B-16
B.2.7	doTranslateFromNative.....	B-17
B.2.8	doTranslateToNative.....	B-17
B.2.9	doXSLTransform.....	B-18
B.2.10	doXSLTransformForDoc.....	B-18
B.2.11	formatDate	B-18
B.2.12	generateGUID	B-19
B.2.13	getApplicationName	B-19
B.2.14	getAttachmentContent.....	B-19
B.2.15	getComponentName	B-20
B.2.16	getComponentInstanceID.....	B-20
B.2.17	getCompositeName	B-20
B.2.18	getCompositeInstanceID	B-20
B.2.19	getCompositeURL	B-21
B.2.20	getContentAsString	B-21
B.2.21	getConversationId	B-21
B.2.22	getCreator	B-21
B.2.23	getCurrentDate.....	B-22
B.2.24	getCurrentDateTime	B-22
B.2.25	getCurrentTime	B-22
B.2.26	getDomainId.....	B-22
B.2.27	getECID	B-23
B.2.28	getElement	B-23
B.2.29	getFaultAsString	B-23
B.2.30	getFaultName.....	B-24
B.2.31	getGroupIdsFromGroupAlias	B-24
B.2.32	getInstanceId	B-24
B.2.33	getNodeValue.....	B-24
B.2.34	getNodes	B-25
B.2.35	getOwnerDocument.....	B-25
B.2.36	getParentComponentInstanceID	B-25
B.2.37	getPreference	B-25
B.2.38	getProcessId.....	B-26
B.2.39	getProcessOwnerId	B-26
B.2.40	getProcessURL	B-26
B.2.41	getProcessVersion.....	B-26

B.2.42	getUserAliasId.....	B-27
B.2.43	getUserIdsFromGroupAlias.....	B-27
B.2.44	setCompositeInstanceTitle	B-27
B.2.45	instanceOf	B-28
B.2.46	integer	B-28
B.2.47	parseEscapedXML	B-28
B.2.48	parseXML.....	B-28
B.2.49	processXQuery	B-29
B.2.50	processXSLT	B-29
B.2.51	processXSLTAttachment	B-29
B.2.52	processXSQL.....	B-30
B.2.53	readBinaryFromFile.....	B-30
B.2.54	readFile	B-30
B.2.55	writeBinaryToFile	B-31
B.2.56	BPEL Extension Functions.....	B-31
B.2.56.1	getLinkStatus.....	B-31
B.2.56.2	getVariableData	B-32
B.2.56.3	getVariableProperty	B-32
B.2.57	Utility Functions	B-33
B.2.57.1	batchProcessActive.....	B-33
B.2.57.2	batchProcessCompleted	B-33
B.2.57.3	format	B-33
B.2.57.4	genEmptyElem.....	B-34
B.2.57.5	getChildElement	B-34
B.2.57.6	getMessage	B-34
B.2.57.7	max-value-among-nodeset.....	B-35
B.2.57.8	min-value-among-nodeset	B-35
B.2.57.9	square-root.....	B-36
B.2.57.10	translateFromNative	B-36
B.2.57.11	translateToNative	B-36
B.2.57.12	translateFromNativeAttachment	B-36
B.2.57.13	translateToNativeAttachment	B-37
B.3	Mediator XPath Extension Functions.....	B-37
B.3.1	getComponentInstanceID.....	B-37
B.3.2	getComponentName	B-37
B.3.3	getCompositeInstanceID	B-38
B.3.4	getCompositeName.....	B-38
B.3.5	getHeader.....	B-38
B.3.6	getECID	B-39
B.3.7	getParentComponentInstanceID	B-39
B.3.8	setCompositeInstanceTitle	B-39
B.4	Advanced Functions.....	B-40
B.4.1	create-nodeset-from-delimited-string.....	B-40
B.4.2	generate-guid.....	B-40
B.4.3	lookupPopulatedColumns	B-40
B.4.4	lookupValue	B-41
B.4.5	lookupValue1M.....	B-41

B.4.6	lookupXRef	B-42
B.4.7	lookupXRef1M	B-42
B.4.8	lookup-xml.....	B-43
B.4.9	markForDelete.....	B-43
B.4.10	populateXRefRow.....	B-44
B.4.11	populateXRefRow1M.....	B-44
B.5	Workflow Service Functions	B-44
B.5.1	clearTaskAssignees.....	B-45
B.5.2	createWordMLDocument.....	B-45
B.5.3	getNotificationProperty	B-45
B.5.4	getNumberOfTaskApprovals	B-46
B.5.5	getPreviousTaskApprover	B-46
B.5.6	getTaskAttachmentByIndex.....	B-46
B.5.7	getTaskAttachmentByName	B-47
B.5.8	getTaskAttachmentContents.....	B-47
B.5.9	getTaskAttachmentsCount.....	B-47
B.5.10	getTaskResourceBundleString.....	B-47
B.5.11	wfDynamicGroupAssign.....	B-48
B.5.12	wfDynamicUserAssign.....	B-49
B.5.13	Identity Service Functions	B-49
B.5.13.1	getDefaultRealmName	B-49
B.5.13.2	getGroupProperty	B-50
B.5.13.3	getManager.....	B-50
B.5.13.4	getReportees	B-50
B.5.13.5	getSupportedRealmNames	B-51
B.5.13.6	getUserProperty.....	B-51
B.5.13.7	getUserRoles.....	B-52
B.5.13.8	getUsersInGroup	B-52
B.5.13.9	isUserInRole	B-53
B.5.13.10	lookupGroup.....	B-53
B.5.13.11	lookupUser	B-53
B.6	Using the XPath Building Assistant.....	B-54
B.6.1	XPath Building Assistant Description	B-54
B.6.2	Starting the XPath Building Assistant	B-54
B.6.3	Using the XPath Building Assistant in Oracle JDeveloper: Step-By-Step Example	B-55
B.6.4	Using the XPath Building Assistant in the XSLT Mapper	B-56
B.6.5	Function Parameter Tool Tips.....	B-58
B.6.6	Syntactic and Semantic Validation.....	B-58
B.6.7	Creating Expressions with Free Form Text and XPath Expressions	B-58
B.7	Creating User-Defined XPath Extension Functions.....	B-59
B.7.1	How to Implement User-Defined XPath Extension Functions	B-62
B.7.1.1	How to Implement Functions for the XSLT Mapper	B-62
B.7.1.2	How to Implement Functions for All Other Components	B-62
B.7.2	How to Configure User-Defined XPath Extension Functions.....	B-63
B.7.3	How to Deploy User-Defined Functions to Runtime	B-65

C Deployment Descriptor Properties

C.1	Introduction to Deployment Descriptor Properties.....	C-1
C.1.1	How to Define Deployment Descriptor Properties	C-1
C.1.2	How to Get the Value of a Preference within a BPEL Process.....	C-3
C.2	Deprecated 10.1.3 Properties.....	C-3

D Understanding Sensor Public Views and the Sensor Actions XSD

D.1	Introduction to Sensor Public Views and the Sensor Actions XSD File.....	D-1
D.2	Sensor Public Views.....	D-1
D.2.1	BPM Schema	D-1
D.2.1.1	BPEL_PROCESS_INSTANCES.....	D-1
D.2.1.2	BPEL_ACTIVITY_SENSOR_VALUES	D-2
D.2.1.3	BPEL_FAULT_SENSOR_VALUES	D-3
D.2.1.4	BPEL_VARIABLE_SENSOR_VALUES.....	D-4
D.3	Sensor Actions XSD File.....	D-5

E Oracle BAM Web Services Operations

E.1	DataObjectOperations10131	E-1
E.1.1	Batch	E-1
E.1.1.1	Request Message.....	E-1
E.1.2	Delete.....	E-2
E.1.2.1	Request Message.....	E-2
E.1.3	Insert	E-2
E.1.3.1	Request Message.....	E-2
E.1.4	Update	E-3
E.1.4.1	Request Message.....	E-3
E.1.5	Upsert	E-3
E.1.5.1	Request Message.....	E-3
E.2	DataObjectOperationsByName.....	E-4
E.2.1	Delete.....	E-4
E.2.1.1	Request Message.....	E-4
E.2.2	Get	E-4
E.2.2.1	Request Message.....	E-5
E.2.3	Insert	E-5
E.2.3.1	Request Message.....	E-5
E.2.4	Update	E-5
E.2.4.1	Request Message.....	E-5
E.2.5	Upsert	E-6
E.2.5.1	Request Message.....	E-6
E.3	DataObjectOperationsByID	E-6
E.3.1	Batch	E-7
E.3.1.1	Request Message.....	E-7
E.3.2	Delete.....	E-7
E.3.2.1	Request Message.....	E-7
E.3.3	Insert	E-8
E.3.3.1	Request Message.....	E-8

E.3.4	Update	E-8
E.3.4.1	Request Message.....	E-8
E.3.5	Upsert	E-9
E.3.5.1	Request Message.....	E-9
E.4	DataObjectDefinition Operations	E-9
E.4.1	Create.....	E-9
E.4.1.1	Request Message.....	E-9
E.4.1.2	Response Message	E-12
E.4.2	Delete.....	E-12
E.4.2.1	Request Message.....	E-12
E.4.2.2	Response Message	E-12
E.4.3	Get.....	E-12
E.4.3.1	Request Message.....	E-12
E.4.3.2	Response Message	E-12
E.4.4	Update.....	E-13
E.4.4.1	Request Message.....	E-13
E.4.4.2	Response Message	E-13
E.5	ManualRuleFire Operations	E-13
E.5.1	FireRuleByName.....	E-13
E.5.1.1	Request Message.....	E-14
E.5.1.2	Response Message.....	E-14

F Oracle BAM Alert Rule Options

F.1	Events.....	F-1
F.1.1	In a specific amount of time	F-1
F.1.2	At a specific time today.....	F-1
F.1.3	On a certain day at a specific time.....	F-1
F.1.4	Every interval between two times.....	F-2
F.1.5	Every date interval starting on certain date at a specific time	F-2
F.1.6	When a report changes	F-2
F.1.7	When a data field changes in data object	F-2
F.1.8	When a data field in a report meets specified conditions.....	F-3
F.1.9	When a data field in a data object meets specified conditions.....	F-4
F.1.10	When this rule is launched	F-5
F.2	Conditions.....	F-5
F.2.1	If it is between two times.....	F-5
F.2.2	If It is between two days	F-5
F.2.3	If it is a particular day of the week.....	F-5
F.3	Actions	F-5
F.3.1	Send a report via email	F-5
F.3.2	Send a message via email	F-6
F.3.3	Send a report via email and escalate to another user after a specific amount of time	F-6
F.3.4	Send a parameterized message.....	F-6
F.3.5	Launch a rule.....	F-10
F.3.6	Launch rule if an action fails	F-10
F.3.7	Delete rows from a Data Object.....	F-10

F.3.8	Call a Web Service	F-11
F.3.9	Run an Oracle Data Integrator Scenario.....	F-11
F.4	Frequency Constraint	F-12

G Oracle BAM ICommand Operations and File Formats

G.1	Summary of Individual Operations	G-1
G.2	Detailed Operation Descriptions	G-3
G.2.1	Clear	G-3
G.2.2	Delete	G-3
G.2.3	Export	G-4
G.2.4	Import	G-10
G.2.5	Rename	G-14
G.3	Format of Command File.....	G-15
G.3.1	Inline Content.....	G-15
G.3.2	Command IDs	G-16
G.3.3	Continue On Error	G-17
G.4	Format of Log File.....	G-17
G.5	Sample Export File.....	G-18
G.6	Regular Expressions	G-18

H Normalized Message Properties

H.1	Oracle BPEL Process Manager Properties	H-1
H.2	Oracle Web Services Addressing Properties.....	H-2

I Oracle User Messaging Service Applications

I.1	Send Message to User Specified Channel	I-1
I.1.1	Overview	I-1
I.1.1.1	Provided Files	I-2
I.1.2	Installing and Configuring SOA and User Messaging Service.....	I-2
I.1.2.1	Updating Addresses in Your LDAP User Profile	I-2
I.1.3	Building the Sample	I-3
I.1.4	Creating a New Application Server Connection.....	I-11
I.1.5	Deploying the Project.....	I-13
I.1.6	Configuring User Messaging Preferences.....	I-14
I.1.7	Testing the Sample.....	I-15
I.1.7.1	Verifying the Execution of Sending the Email	I-16
I.2	Send Email with Attachments.....	I-16
I.2.1	Overview	I-16
I.2.1.1	Provided Files	I-17
I.2.2	Installing and Configuring SOA and User Messaging Service.....	I-17
I.2.2.1	Updating Addresses in Your LDAP User Profile	I-17
I.2.3	Running the Pre-Built Sample	I-18
I.2.4	Testing the Sample.....	I-20
I.2.4.1	Verifying the Execution	I-20
I.2.5	Building the Sample	I-20

1.2.6	Creating a New Application Server Connection.....	I-34
-------	---	------

Index

Part

Preface

This manual describes how to use Oracle SOA Suite.

This preface contains the following topics:

- [Audience](#)
- [Documentation Accessibility](#)
- [Related Documents](#)
- [Conventions](#)

Audience

This manual is intended for anyone who is interested in developing applications with Oracle SOA Suite.

Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible to all users, including users that are disabled. 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.

Deaf/Hard of Hearing Access to Oracle Support Services

To reach Oracle Support Services, use a telecommunications relay service (TRS) to call Oracle Support at 1.800.223.1711. An Oracle Support Services engineer will handle technical issues and provide customer support according to the Oracle service request process. Information about TRS is available at

<http://www.fcc.gov/cgb/consumerfacts/trs.html>, and a list of phone numbers is available at <http://www.fcc.gov/cgb/dro/trsphonebk.html>.

Related Documents

For more information, see the following Oracle resources:

- *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite*

Printed documentation is available for sale in the Oracle Store at

<http://oraclestore.oracle.com/>

To download free release notes, installation documentation, white papers, or other collateral, visit the Oracle Technology Network (OTN). You must register online before using OTN; registration is free and can be done at

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

To download Oracle BPEL Process Manager documentation, technical notes, or other collateral, visit the Oracle BPEL Process Manager site at Oracle Technology Network (OTN):

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

If you have a username and password for OTN, then you can go directly to the documentation section of the OTN web site at

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

See the *Business Process Execution Language for Web Services Specification*, available at the following URL:

<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbizspec/html/bpel1-1.asp>

See the *XML Path Language (XPath) Specification*, available at the following URL:

<http://www.w3.org/TR/1999/REC-xpath-19991116>

See the *Web Services Description Language (WSDL) 1.1 Specification*, available at the following URL:

<http://www.w3.org/TR/wsdl>

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
<i>italic</i>	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.

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

Part I

Introduction to Oracle SOA Suite

This part provides an introduction to Oracle SOA Suite and developing SOA composite applications.

This part contains the following chapters:

- [Chapter 1, "Introduction to SOA Composite Applications"](#)
- [Chapter 2, "Overview of SOA Component Editors"](#)
- [Chapter 3, "Introduction to the SOA Sample Application"](#)
- [Chapter 4, "Introduction to the Functionality of the SOA Composite Editor"](#)

Introduction to SOA Composite Applications

An SOA composite application is an assembly of services, service components, references, and wires designed and deployed together to meet a business need. This chapter provides a high-level introduction to the various components that together form an SOA composite application.

This chapter includes the following sections:

- [Section 1.1, "Introduction to Oracle SOA Suite"](#)
- [Section 1.2, "Introduction to SOA Composite Applications"](#)
- [Section 1.3, "Introduction to SCA Technologies"](#)
- [Section 1.4, "Learning Oracle SOA Suite"](#)

1.1 Introduction to 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 must 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 SOA to deliver on these requirements by overcoming the complexity of their application and IT environments.

SOA provides an enterprise architecture that supports building connected enterprise applications. SOA facilitates the development of enterprise applications as modular business web services that can be easily integrated and reused, creating a truly flexible, adaptable IT infrastructure.

Oracle SOA Suite provides a complete set of service infrastructure components for designing, deploying, and managing composite applications. Oracle SOA Suite enables services to be created, managed, and orchestrated into composite applications and business processes. Composites enable you to easily assemble multiple technology components into one SOA composite application. Oracle SOA Suite 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

- Activity monitoring
- Web services management and security
- Business rules
- Events framework

Oracle SOA Suite puts a strong emphasis on standards and interoperability. Among the standards it leverages are:

- Service Component Architecture (SCA) assembly model
Provides the service details and their interdependencies to form composite applications. SCA enables you to represent business logic as *reusable* service components that can be easily integrated into any SCA-compliant application. The resulting application is known as an SOA composite application. The specification for the SCA standard is maintained by the Organization for the Advancement of Structured Information Standards (OASIS) through the Open Composite Services Architecture (CSA) Member Section:
<http://www.oasis-open.org>
- Service Data Objects (SDO)
Specifies a standard data method and can modify business data regardless of how it is physically accessed. Knowledge is not required about how to access a particular back-end data source to use SDO in an SOA composite application. Consequently, you can use static or dynamic programming styles and obtain connected and disconnected access.
- Business Process Execution Language (BPEL)
Provides enterprises with an industry standard for business process orchestration and execution. Using BPEL, you design a business process that integrates a series of discrete services into an end-to-end process flow. This integration reduces process cost and complexity.
- XSL Transformations (XSLT)
Processes XML documents and transforms document data from one XML schema to another.
- Java Connector Architecture (JCA)
Provides a Java technology solution to the problem of connectivity between the many application servers in Enterprise Information Systems (EIS).
- Java Messaging Service (JMS)
Provides a messaging standard that allows application components based on the Java 2 Platform, Enterprise Edition (Java EE) to access business logic distributed among heterogeneous systems.
- Web Services Description Language (WSDL) file
Provides the entry points into an SOA composite application. The WSDL file provides a standard contract language and is central for understanding the capabilities of a service.
- Simple Object Access Protocol (SOAP)
Provides the default network protocol for message delivery.

For more information about standards, see the following:

- [Section 1.3, "Introduction to SCA Technologies"](#)

- Section 2.1, "Introduction to the SOA Composite Editor" for additional details about these key building blocks
- The following URL for SCA and SDO specifications and related material:
<http://www.osoa.org>

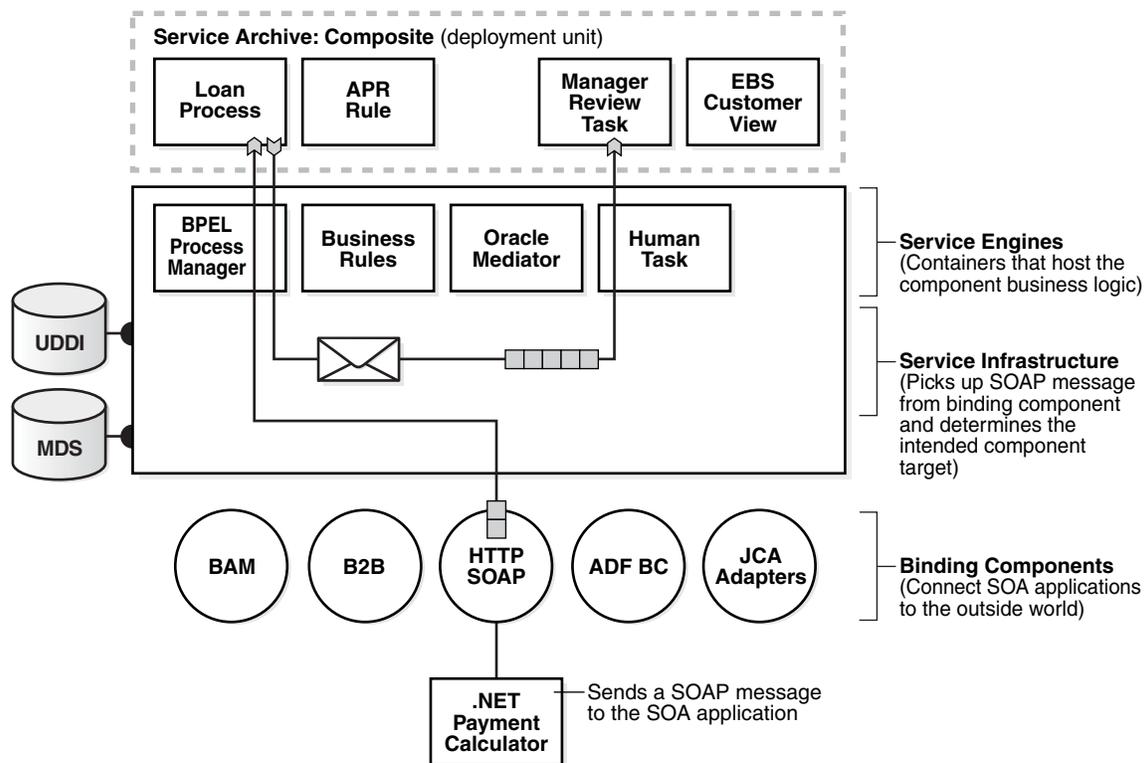
1.2 Introduction to SOA Composite Applications

A composite is an assembly of services, service components, wires, and references designed and deployed together in a single application. The composite processes the information described in the messages.

Figure 1-1 describes the operability of an SOA composite application using SCA technology. In this example, an external application (.NET payment calculator) initiates contact with the SOA composite application.

For more information about descriptions of the tasks that services, references, service components, and wires perform in an application, see Section 1.3, "Introduction to SCA Technologies."

Figure 1-1 Introduction to an SOA Composite Application



The .NET payment calculator is an external application that sends a SOAP message to the SOA application to initiate contact. The Service Infrastructure picks up the SOAP message from the binding component and determines the intended component target. The BPEL service engine receives the message from the Service Infrastructure for processing by the BPEL Loan Process application and posts the message back to the Service Infrastructure after completing the processing.

Table 1-1 describes the operability of the SOA composite application shown in Figure 1-1.

1.3 Introduction to SCA Technologies

SCA is the executable model for the assembly of service components into composite applications. SCA provides a programming model for the following:

- Creating service components written with a wide range of technologies, including programming languages such as Java, BPEL, C++, and declarative languages such as XSLT. The use of specific programming languages and technologies (including web services) is not required with SCA.
- Assembling the service components into an SOA composite application. In the SCA environment, service components are the building blocks of applications.

SCA lets you describe the details of a service and how services and service components interact by providing a model for assembling distributed groups of service components into an application. Composites are used to group service components and wires are used to connect service components. SCA aims to remove middleware concerns from the programming code by applying infrastructure concerns declaratively to compositions, including security and transactions.

The key benefits of SCA include the following:

- Loose coupling
Service components integrate with other service components without needing to know how other service components are implemented.
- Flexibility
Service components can easily be replaced by other service components.
- Services invocation
Services can be invoked either synchronously or asynchronously.
- Productivity
Service components are easily integrated to form an SOA composite application.
- Easy Maintenance and Debugging
Service components can be easily maintained and debugged when encountered an issue.

[Figure 1–2](#) provides an example of a composite that includes an inbound service binding component, a BPEL process service component (named Account), a business rules service component (named AccountRule), and two outbound reference binding components. The details of this composite are stored in the composite.xml file.

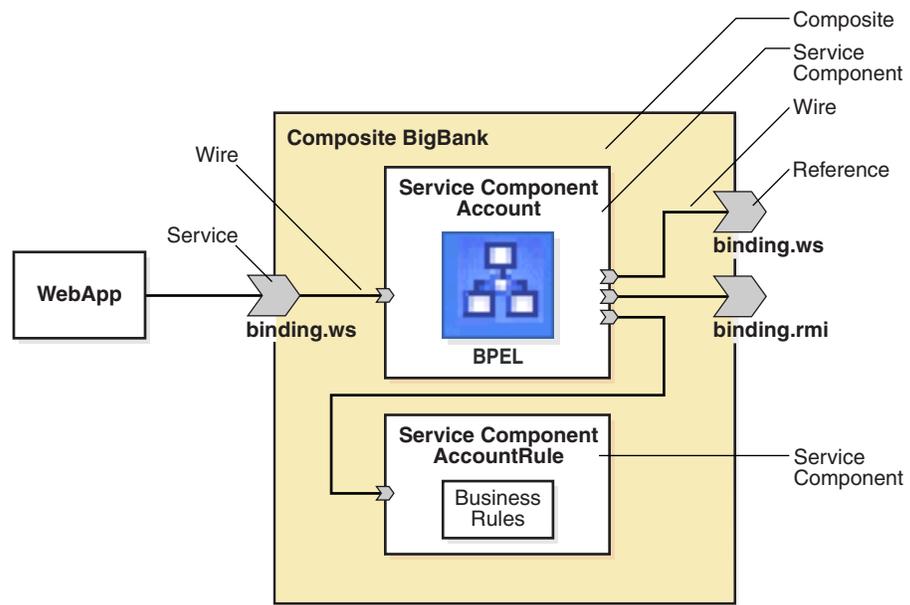
Figure 1-2 Composite

Table 1-1 describes the operability of the SOA composite application shown in Figure 1-1. References are made to sections that provide additional details.

Table 1–1 Introduction to an SOA Composite Application Using SCA Technologies

Part	Description	Example of Use in Figure 1–1	See Section
Binding Components	Establishes the connectivity between a SOA composite and the external world. There are two types: <ul style="list-style-type: none"> ▪ <i>Service</i> binding components provide an entry point to the SOA composite application. ▪ <i>Reference</i> binding components enable messages to be sent from the SOA composite application to external services. 	The SOAP binding component <i>service</i> : <ul style="list-style-type: none"> ▪ Advertises its capabilities in the WSDL file. ▪ Receives the SOAP message from the .NET application. ▪ Sends the message through the policy infrastructure for security checking. ▪ Translates the message to a normalized message (an internal representation of the service's WSDL contract in XML format). ▪ Posts the message to the Service Infrastructure. <p>An example of a binding component <i>reference</i> in Figure 1–1 is the Loan Process application.</p>	Section 1.3.1, "Binding Components"
Service Infrastructure	Provides internal message transport	The Service Infrastructure: <ul style="list-style-type: none"> ▪ Receives the message from the SOAP binding component service. ▪ Posts the message for processing to the BPEL process service engine first and the human task service engine second. 	Section 1.3.2, "Service Infrastructure"
Service Engines (containers hosting service components)	Host the business logic or processing rules of the service components. Each service component has its own service engine.	The BPEL service engine: <ul style="list-style-type: none"> ▪ Receives the message from the Service Infrastructure for processing by the BPEL Loan Process application. ▪ Posts the message to the Service Infrastructure after completing the processing. 	Section 1.3.3, "Service Engines and Service Components"
UDDI and MDS	The MDS (Metadata Service) repository stores descriptions of available services. The UDDI advertises these services, and enables discovery as well as dynamic binding at runtime.	The SOAP service used in this composite application is stored in the MDS and can also be published to UDDI.	<i>Oracle Fusion Middleware Getting Started for Oracle SOA Suite</i>
SOA Archive: Composite (deployment unit)	The deployment unit that describes the composite application.	The SOA archive (SAR) of the composite application is deployed to the Service Infrastructure.	Section 1.3.4, "Deployed Service Archives"

1.3.1 Binding Components

Binding components establish the connection between a SOA composite and the external world. There are two types of binding components:

- **Services**

Provide the outside world with an entry point to the SOA composite application. The WSDL file of the service advertises its capabilities to external applications. These capabilities are used for contacting the SOA composite application components. The binding connectivity of the service describes the protocols that can communicate with the service, for example, SOAP/HTTP or a JCA adapter.

- **References**

Enable messages to be sent from the SOA composite application to external services in the outside world.

Table 1–2 lists and describes the web services provided by Oracle SOA Suite.

Table 1–2 Web Services Provided by Oracle SOA Suite

Web Services	Description
SOAP over HTTP	For connecting to standards-based services using SOAP over HTTP.
JCA Adapters	For integrating services and references with technologies (for example, databases, file systems, FTP servers, messaging: JMS, IBM WebSphere MQ, and so on) and applications (Oracle E-Business Suite, PeopleSoft, and so on). This includes AQ Adapter, Database Adapter, File Adapter, FTP Adapter, JMS Adapter, MQ Adapter, and Socket Adapter.
B2B binding component	For browsing B2B metadata in the MDS repository and selecting document definitions.
ADF-BC Service	For connecting Oracle Application Development Framework (ADF) applications using SDO with the SOA platform.
Oracle Applications	For integrating Oracle Application Adapter with Oracle Applications.
BAM Adapter	For integrating Java EE applications with Oracle BAM Server to send data and also used as a reference binding component in an SOA composite application.
EJB Service	For integrating SDO parameters with Enterprise JavaBeans.

Note: Business events provide an alternative to using the direct service invocation of the WSDL file contract. Business events are messages sent as the result of an occurrence or situation. When a business event is published, other applications can subscribe to it.

1.3.2 Service Infrastructure

The Service Infrastructure provides the internal message routing infrastructure capabilities for connecting components and enabling data flow:

- Receives messages from the service providers or external partners through SOAP services or adapters
- Sends the message to the appropriate service engine
- Receives the message back from the service engine and sends it to any additional service engines in the composite or to a reference binding component based on the wiring

1.3.3 Service Engines and Service Components

Service components are the building blocks that you use to construct an SOA composite application. Service engines are containers that host the business logic or processing rules of these service components. Service engines process the message information received from the Service Infrastructure.

The following service components are available. There is a corresponding service engine of the same name for each service component. All service engines can interact together in a single composite.

- BPEL process
For process orchestration and storage of synchronous or asynchronous process. You design a business process that integrates a series of business activities and services into an end-to-end process flow.
- Business rules
For designing a business decision based on rules.
- Human task
For modeling a workflow that describes the tasks for users or groups to perform as part of an end-to-end business process flow.
- Mediator
For routing events (messages) between different components.

1.3.4 Deployed Service Archives

The SAR is a SOA archive deployment unit. The SAR file is deployed to the Service Infrastructure. The SAR packages service components such as BPEL processes, business rules, human tasks, and mediator routing services into a single application. The SAR file is analogous to the BPEL suitcase archive of previous releases, but at the higher composite level and with any additional service components that your application includes (for example, human tasks, business rules, and mediator routing services).

1.3.5 Wires

Wires enable you to graphically connect the following components in a single SOA composite application for message communication:

- Services to service components
- Service components to other service components
- Service components to references

1.4 Learning Oracle SOA Suite

In addition to this developers guide, Oracle also offers the following resources to help you learn how you can best use Oracle SOA Suite in your applications:

- Getting Started and Tutorials: The *Oracle Fusion Middleware Getting Started with Oracle SOA Suite* guide introduces you to Oracle SOA Suite, its components, and provides you with a high-level understanding of what you can accomplish with the suite. The Oracle Fusion Middleware Tutorial for Running and Building an Application with Oracle SOA Suite tutorial describe the step-by-step instructions for running an application developed with Oracle SOA Suite and how to build the SOA elements of the Fusion Order Demo application respectively. Also, you could refer to the Oracle SOA Suite section of the Oracle Fusion Middleware 11g Release 1 documentation library for additional documentation.
- Cue Cards in Oracle JDeveloper: Oracle JDeveloper cue cards provide step-by-step support for the application development process using Oracle SOA Suite. They are designed to be used either with the included examples and a sample schema, or with your own data. Cue cards also include topics that provide more detailed background information, viewlets that demonstrate how to complete the steps in the card. Cue cards provide a fast, easy way to become familiar with the basic

features of Oracle SOA Suite, and to work through a simple end-to-end task. In Oracle JDeveloper, click **Help, Cue Cards** to access the cue cards.

- http://www.oracle.com/technology/sample_code/products/soa: The SOA OTN provides access to various use case samples for Oracle SOA Suite and its components.

Overview of SOA Component Editors

This chapter provides an overview of all the component editors and designers, which are used to develop an SOA composite application.

This chapter includes the following sections:

- [Section 2.1, "Introduction to the SOA Composite Editor"](#)
- [Section 2.2, "Introduction to the Oracle BPEL Designer"](#)
- [Section 2.3, "Introduction to the Oracle Mediator Editor"](#)
- [Section 2.4, "Introduction to the Human Task Editor"](#)
- [Section 2.5, "Introduction to the Business Rules Designer"](#)
- [Section 2.6, "Introduction to Oracle Enterprise Manager"](#)

2.1 Introduction to the SOA Composite Editor

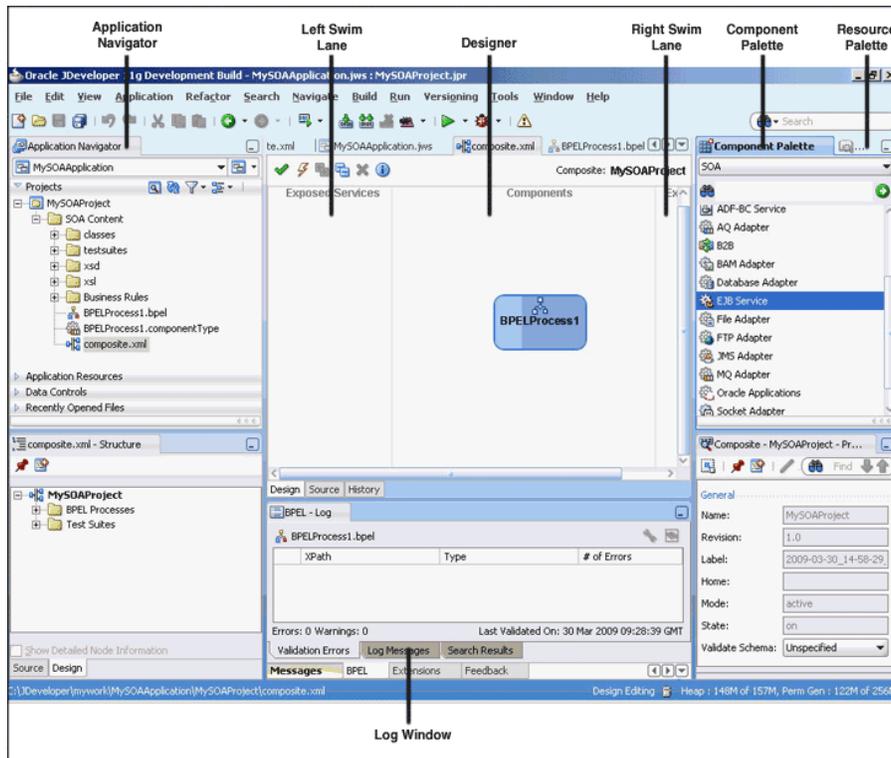
The SOA Composite Editor enables you to create, edit, and deploy services, and also to assemble them in a composite application, all from a single location. These components are integrated together into one application and communicate with the outside world through binding components such as web services and JCA adapters.

The SOA Composite Editor enables you to use either of two approaches for designing SOA composite applications:

- The top-down approach of building a composite application puts interfaces first and implementation next. For example, you first add BPEL processes, human tasks, business rules, and mediator routing services components to an application, and later define the specific content of these service components.
- The bottom-up approach takes existing implementations of service components and wraps them with web service interfaces for assembly into a composite application. For example, you first create and define the specific content of BPEL processes, human tasks, business rules, and mediator routing services components, and later create an SOA composite application to which you add these service components.

The SOA Composite Editor appears as shown in [Figure 2-1](#).

Figure 2–1 SOA Composite Editor



The main sections of the SOA Composite Editor are described in the following list:

2.1.1 Application Navigator

Displays the key files for the specific service components included in your SOA project:

- A `composite.xml` file that is automatically created when you create a SOA project. This file describes the entire composite assembly of services, service components, references, and wires.
- The business rules service component file (`rules_name.decs`). Additional business rules files display under the **Oracle > rules** subfolder (`rules_name.rules`).
- The mediator service component file (`mediator_name.mplan`).
- The BPEL process service component files (`process_name.bpel` and `process_name.wsdl`).
- The human task service component files (`task_name.task`).
- The `componentType` file that describes the services and references for each service component. This file ensures that the wiring you create between components works.
- Additional subfolders for class files, XSDs (schemas), and XSLs (transformations).

You can drag and drop components and service adapters from the Components Palette window to the Designer window. When you drop a service component into the Designer window, it starts a property editor for configuring that service component. For example, when you drop a Mediator component into the Designer window, this also opens the Mediator editor window that enables configure the Mediator.

To edit the configuration of an existing component in the Designer window, double-click the component to re-open the editor.

2.1.2 Designer

You drag service components, services, and references into the composite in the designer. When you drag and drop a service component into the Designer window, a corresponding property editor is invoked for performing configuration tasks related to that service component. For example, when you drag and drop the Mediator component into Designer, then the Mediator Editor window is displayed that enables you to configure the Mediator component.

For all subsequent editing sessions, you double-click these service components to invoke their editors.

2.1.3 Left Swim Lane (Exposed Services)

The left swim lane is for services providing an entry point to the SOA composite application, such as a web service or JCA adapters.

2.1.4 Right Swim Lane (External References)

The right swim lane is for references that send messages to external services in the outside world, such as web services and JCA adapters.

2.1.5 Component Palette

Contains the various resources that you can use in a SOA composite. It contains the following service components and adapters:

- Service components
 - Displays the BPEL Process, business rule, human task, and mediator service that can be dragged and dropped into the designer.
- Service adapters
 - Displays the JCA adapter (AQ, file, FTP, Database, JMS, MQ, Oracle Applications, Oracle BAM, and EJB Service), B2B binding component, SDO binding component, and web service binding component that can be dragged into the left or right swim lanes.

If the Resource Catalog does not display, select **Component Palette** from the **View** main menu.

2.1.6 Resource Palette

Provides a single dialog from which you can browse both local and remote resources. For example, you can access.

- Shared local application metadata such as schemas, WSDLs, event definitions, business rules, and so on.
- WSIL browser functionality that uses remote resources that can be accessed through an HTTP connection, file URL or Application Server connection.
- Remote resources that are registered in a UDDI (Universal Description, Discover and Integration) registry.

If the Resource Catalog does not display, then select **Resource Palette** from the **View** main menu.

You select these resources for the SOA composite application through the SOA Resource Browser dialog. This dialog is accessible through a variety of methods. For example, when you select the WSDL file to use with a service binding component or a mediator service component or select the schema file to use in a BPEL process, the SOA Resource Browser dialog appears. Click Resource Palette at the top of this dialog to access available resources.

2.1.7 Log Window

Displays messages about application compilation, validation, and deployment.

2.1.8 Property Inspector

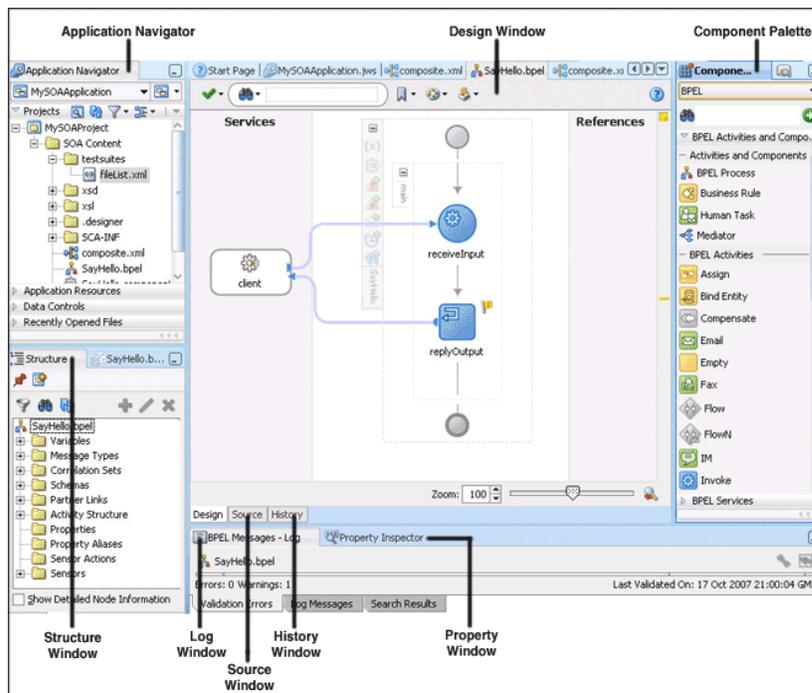
Displays properties for the selected service component, service, or reference.

If the Property Inspector does not display, select Property Inspector from the **View** main menu.

For more information about the SOA Composite Editor, see [Chapter 4, "Introduction to the Functionality of the SOA Composite Editor"](#) and *Oracle Fusion Middleware Getting Started with Oracle SOA Suite*.

2.2 Introduction to the Oracle BPEL Designer

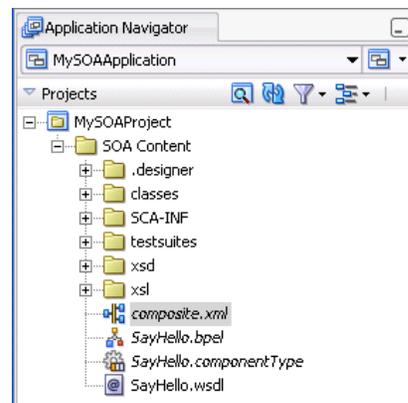
You can create a BPEL process service component in the SOA composite application of Oracle JDeveloper and then design it by using the BPEL Designer, which is displayed, when you double-click a BPEL process in the SOA Composite Editor. [Figure 2-2](#) shows the BPEL Designer along with Application Navigator, Structure, Component Palette, and Messages windows.

Figure 2–2 Oracle BPEL Designer

Each section of this view enables you to perform specific design and deployment tasks. The main sections of the BPEL Designer are described in the following list:

2.2.1 Application Navigator

The Application Navigator displays the process files. [Figure 2–3](#) shows the files that appear under the **SOA Content** folder when you first create a SOA project in Oracle JDeveloper (in this example, named **MySOAProject** inside an application named **MySOAApplication**, **SayHello** is the name of the BPEL process). An application can contain one or more projects. Each project can only contain one composite. But each composite can have multiple components of either the same type or different types (BPEL process, Oracle Mediator, human workflow, and business rules).

Figure 2–3 Application Navigator

[Table 2–1](#) describes these initial process files.

Table 2–1 Initial Process Files

File	Description
composite.xml	The file that describes the entire SOA composite. For more information about this file, see Section 2.1, "Introduction to the SOA Composite Editor"
SayHello.bpel	The source file, which, depending upon the process type you selected, initially contains a minimal set of activities (if you selected to create an asynchronous process, then receive and invoke activities appear). You add syntax to this file when you drag activities, create variables, create partner links, and so on.
SayHello.component Type	The file that describes the services and references for each service component.
SayHello.wsdl	The Web Services Description Language (WSDL) client interface, which defines the input and output messages for this BPEL process flow, the supported client interface and operations, and other features. This functionality enables the BPEL process flow to be called as a service.

As you design the BPEL process service component, additional files, folders, and elements can appear in the Application Navigator.

Note: If you want to learn more about the Application Navigator, place the cursor in this section and press **F1** to display online Help.

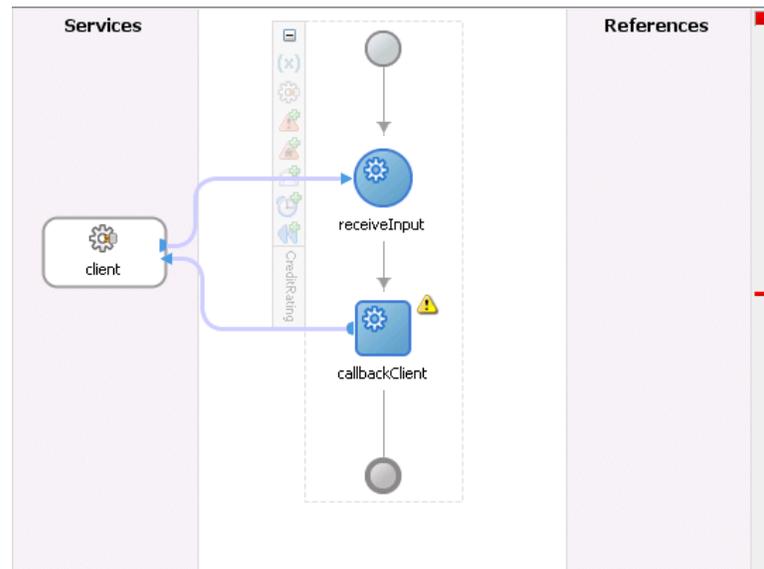
2.2.2 Design Window

The **Design** window provides a visual view of the BPEL process service component that you design. This view displays when you perform one of the following actions:

- Double-click the **.bpel** file name in the Application Navigator.
- Double-click the BPEL process component in the SOA Composite Editor.
- Click the **Design** tab at the bottom of the window with the **.bpel** file selected.

[Figure 2–4](#) shows the activities automatically created with an asynchronous BPEL process service component. You add to the BPEL process service component by dragging and dropping activities, creating variables, creating partner links, and so on.

Figure 2–4 Design (After Creation of an Asynchronous BPEL Process Service Component)

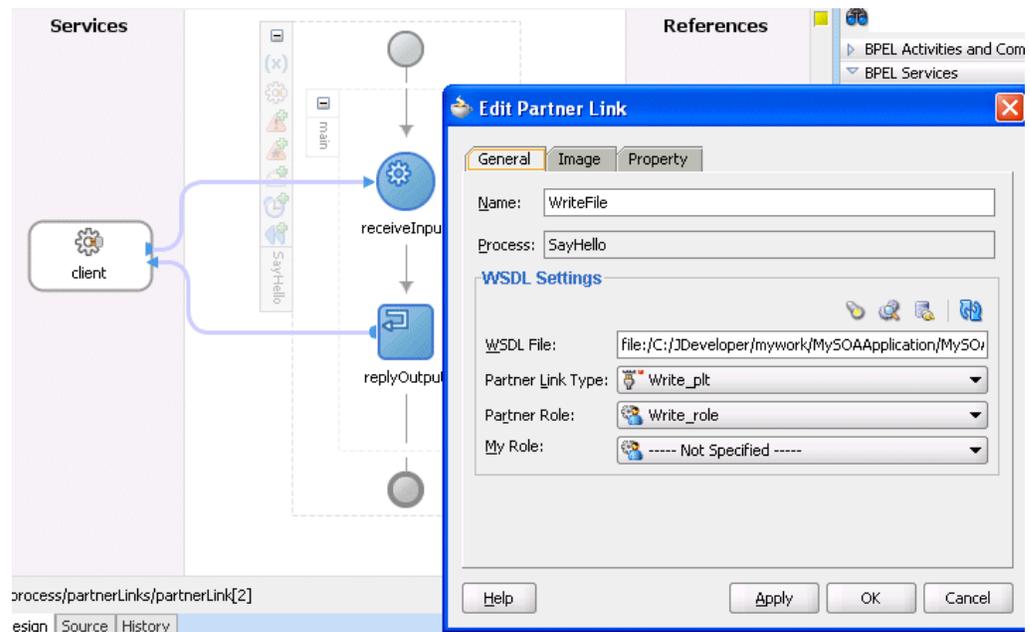


2.2.3 Source Window

Click **Source** at the bottom to view the syntax inside the BPEL process service component files. As you drag activities and partner links, and perform other tasks, the **Source** view and **Design** view stay synchronized. Changes in one are reflected in the other immediately.

For example, [Figure 2–5](#) shows the property sheet as it is being edited.

Figure 2–5 WriteFile Partner Link Icon and Property Sheet



Click **Source** at the bottom of the window. [Figure 2–6](#) shows part of the **Source** of a **.bpel** file. Details about the **WriteFile** partner link you created appear in the file.

Figure 2–6 Source View of a .bpel File

```

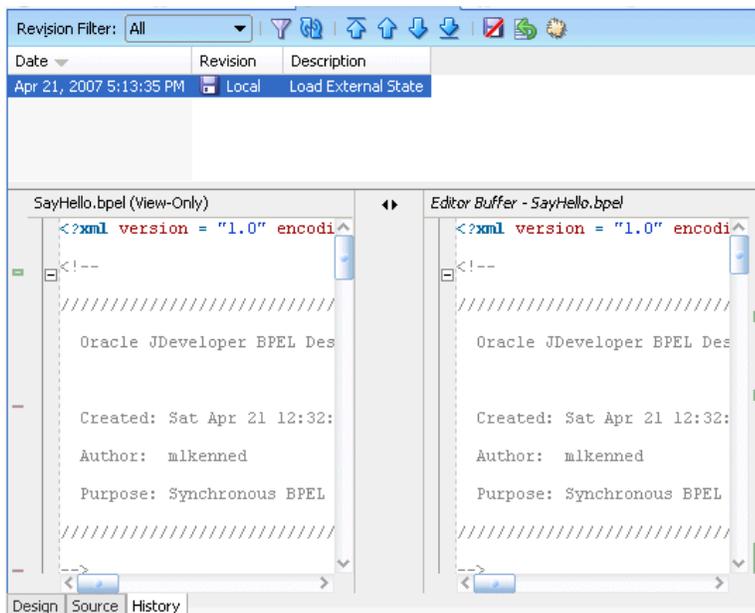
<!--
  //////////////////////////////////////
  PARTNERLINKS
  List of services participating in this BPEL process
  //////////////////////////////////////
-->
<partnerLinks>
  <!--
    The 'client' role represents the requester of this service. It is
    used for callback. The location and correlation information associat
    with the client role are automatically set using WS-Addressing.
  -->
  <partnerLink name="client" partnerLinkType="client:SayHello" myRole="S
  <partnerLink name="WriteFile" partnerRole="Write_role"
    partnerLinkType="nsl:Write_plt"/>
</partnerLinks>

```

For more information about the types of syntax that appear in BPEL process service component files, see [Section II, "Using the BPEL Process Service Component."](#)

2.2.4 History Window

Click **History** at the bottom to perform such tasks as viewing the revision history of a file and viewing read-only and editable versions of a file side-by-side. [Figure 2–7](#) shows the **History** view for a BPEL file.

Figure 2–7 History View

Note: If you want to learn more about the **History** view, place the cursor in this section and press **F1** to display online Help.

2.2.5 Component Palette

Activities are the building blocks of the BPEL process service component. The **BPEL Activities** selection of the Component Palette displays a set of activities that you drag into the **Design** window of the BPEL process service component. The Component Palette is context-aware and only displays those pages relevant to the state of the **Design** window. **BPEL Activities** or **BPEL Services** are nearly always visible. However, if you are designing a transformation in a **transform** activity, the Component Palette only displays selections relevant to that activity, such as **String Functions**, **Mathematical Functions**, and **Node-set Functions**.

Figure 2–8 shows the **BPEL Activities** selection of the Component Palette. This list enables you to select activities to drag into your BPEL process service component.

Figure 2–8 Component Palette - BPEL Activities

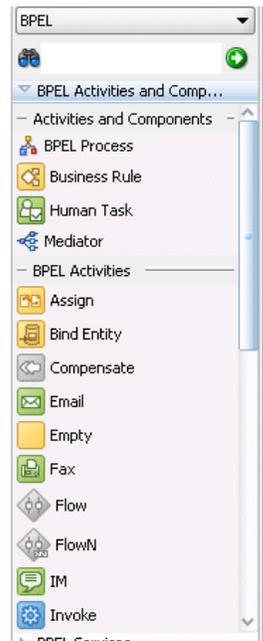
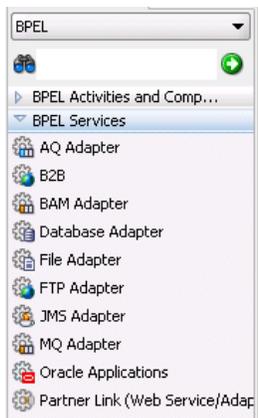


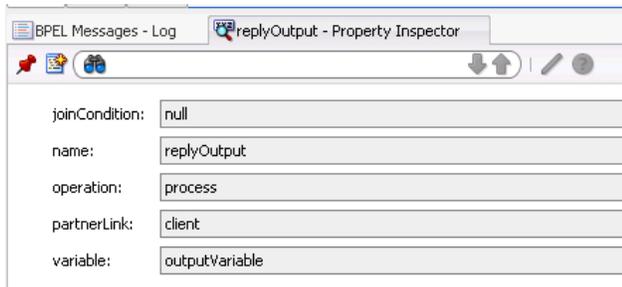
Figure 2–9 shows the **BPEL Services** selection of the Component Palette. This list enables you to drag adapters, partner links, or decision services into your BPEL process service component.

Figure 2–9 Component Palette - Services

Note: If you want to learn more about the Component Palette, place the cursor in this section and press **F1** to display online Help.

2.2.6 Property Inspector

The Property Inspector enables you to view details about an activity. Single-click an activity in the **Design** window. For example, single-clicking the **replyOutput receive** activity displays the information shown in [Figure 2–10](#).

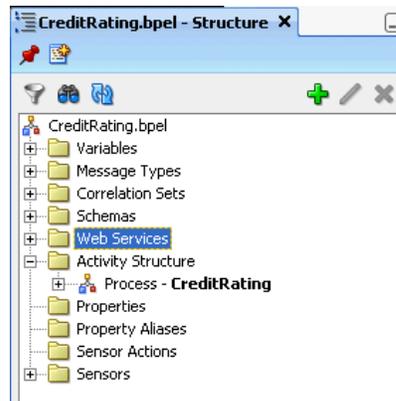
Figure 2–10 Property Inspector

2.2.7 Structure Window

The Structure window offers a structural view of the data in the BPEL process service component currently selected in the **Design** window. You can perform a variety of tasks from this section, including:

- Importing schemas
- Defining message types
- Managing (creating, editing, and deleting) elements such as variables, aliases, correlation sets, partner links, and sensors
- Creating activities in the BPEL process flow sequence using the **Structure** window

[Figure 2–11](#) shows the Structure window.

Figure 2–11 Structure Window (Expanded)**Notes:**

- If you want to learn more about the Structure window, place the cursor in this section and press **F1** to display online Help.
- Do not import two schema files with the same name into a BPEL process service component. Ensure that the files have unique names.

2.2.8 Log Window

The Log window displays messages about the status of validation and compilation. If deployment is unsuccessful, messages appear that describe the type and location of the error.

Note: If you want to learn more about the Log window, place the cursor in this section and press **F1** to display online Help.

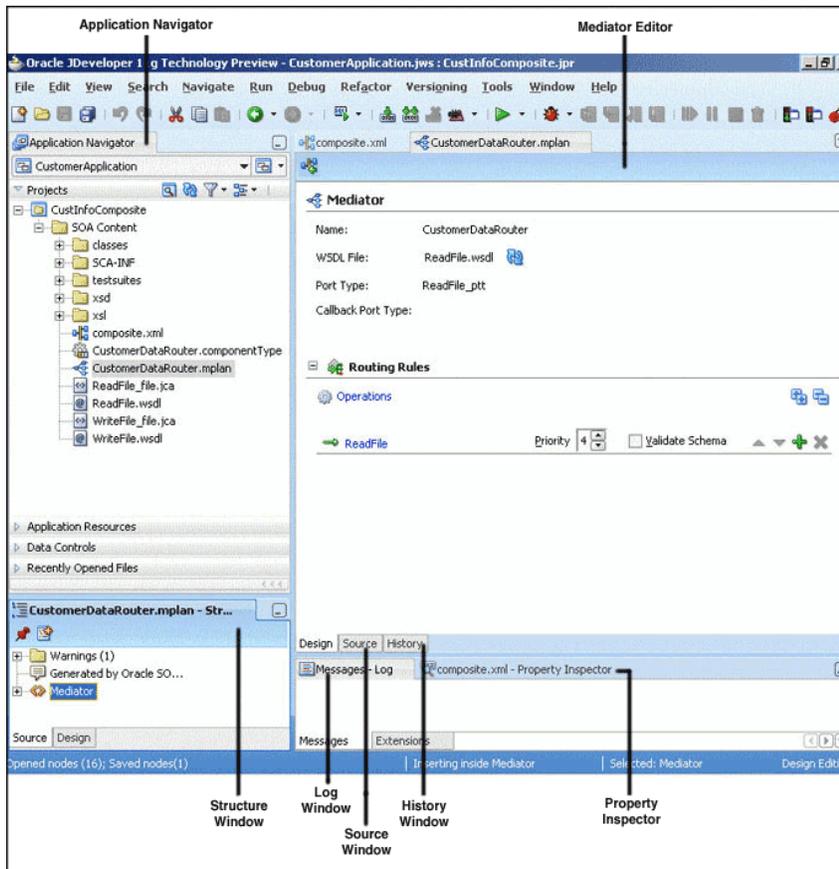
For more information about BPEL, refer to [Part II, "Using the BPEL Process Service Component"](#).

2.3 Introduction to the Oracle Mediator Editor

You can create a Mediator in the SOA composite application of Oracle JDeveloper and then design it by using the Mediator Editor, which is displayed when you double-click a Mediator in SOA Composite Editor.

[Figure 2–12](#) shows the Mediator Editor along with Application Navigator, Structure, and Messages windows.

Figure 2–12 Mediator Editor Window

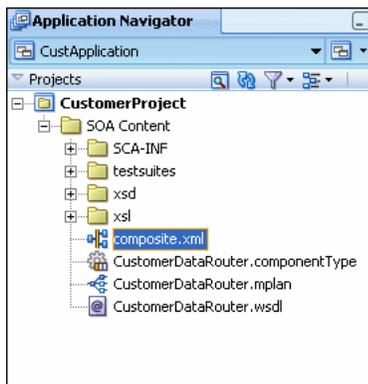


Each section in the Mediator Editor window enables you to perform specific design and deployment tasks. The main sections of the Mediator Editor are described in the following list:

2.3.1 Application Navigator

The Application Navigator shown in the upper left part of [Figure 2–12](#) displays the Mediator files. [Figure 2–13](#) shows the files that appear under the SOA Content folder when you create a Mediator in a SOA Composite application.

Figure 2–13 Mediator Files in Application Navigator



As shown in [Figure 2-13](#), a SOA Composite application consists of the following Mediator files:

- `Composite.xml`: The file that describes the entire SOA composite application.
- `.componentType`: The `.componentType` file describes the services and references for a service component.
- `.mplan`: The `.mplan` file contains Mediator metadata.
- `.wsdl`: A Web Service Description File (WSDL) file specifies how other services call a Mediator. A WSDL file defines the input and output messages and operations of a Mediator.

2.3.2 Mediator Editor

The Mediator Editor provides a visual view of the Mediator that you have created. This view is displayed when you perform one of the following actions:

- Double-click a Mediator in the SOA Composite Editor.
- Double-click the `.mplan` file name in the Application Navigator.

2.3.3 Source View

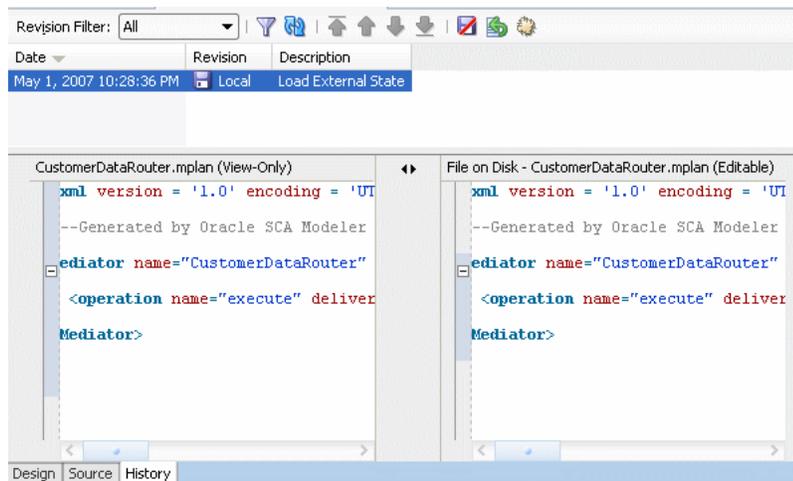
The Source View enables you to view the source code of a Mediator. Click Source at the bottom of the Design window, to view to source code. The code in the source view is immediately updated to reflect the changes in a Mediator.

The following example shows a sample Mediator source code:

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<!--Generated by Oracle SCA Modeler version 1.0 at [4/16/07 10:05 PM].-->
<Mediator name="CustomerDataRouter"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://xmlns.oracle.com/sca/1.0/mediator"/>
```

2.3.4 History Window

The History window enables you to perform tasks as viewing the revision history of a file and viewing read-only and editable versions of a file side-by-side. Click History at the bottom of the Design window, to open the History window. [Figure 2-14](#) shows the History view for a Mediator file.

Figure 2–14 History Window

2.3.5 Property Inspector

The Property Inspector enables you to view details about Mediator properties.

2.3.6 Structure Window

The Structure Window provides a structural view of the data of a Mediator.

2.3.7 Log Window

The Log Window displays messages about the status of validation and compilation.

For more information about Mediator, refer to [Part III, "Using the Oracle Mediator Service Component"](#).

2.4 Introduction to the Human Task Editor

You can create a human task service component in the SOA composite application of Oracle JDeveloper and then design it by using the Human Task Editor, which is displayed when you double-click a human task in the SOA Composite Editor.

The Human Task Editor consists of the following main sections shown in [Figure 2–15](#). These sections enable you to design the metadata of a human task.

Figure 2–15 Human Task Editor

The screenshot shows the Human Task Editor interface with the following sections and fields:

- Task Title Section:**
 - Title: Text and XPath (selected), Approval Required for Order Id: <%/task:task/task:payload/task:orderId%>
 - Description: (empty text box)
 - Outcomes: APPROVE,REJECT
 - Category: By Expression
 - Priority: 3 (Normal)
 - Owner: User
 - Static: (checkbox)
- Parameters Section:**
 - Table with columns: Name, Element or Type, Editable
 - Row: orderId, {http://www.w3.org/2001/XMLSchema}string
- Assignment and Routing Policy Section:**
 - Assignment and Routing Policy
 - Buttons: Make Parallel, Make Serial, Edit
- Expiration and Escalation Policy Section:**
 - Expiration and Escalation Policy
- Notification Settings Section:**
 - Notification Settings
- Advanced Settings Section:**
 - Advanced Settings
- Annotations Section:**
 - Annotations

The main sections of the Human Task Editor are described in the following list:

2.4.1 Task Title

This section enables you to specify details such as the task title, description, task outcomes, task category, task priority, and task owner.

2.4.2 Parameters

This section enables you to define the structure (message elements) of the task payload (the data in the task) defined in the XSD file. [Figure 2–16](#) describes the **Parameters** section of the Human Task Editor.

Figure 2–16 Human Task Editor — Parameters Section

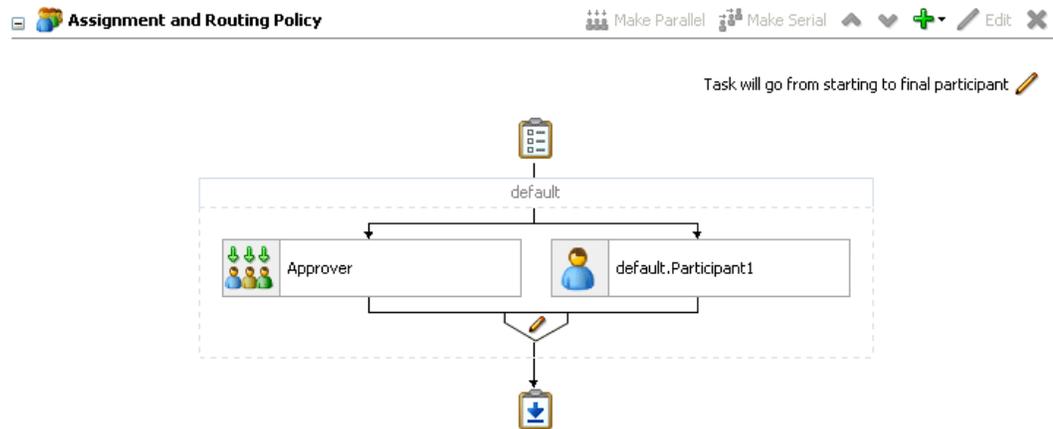
Name	Element or Type	Editable
orderId	{http://www.w3.org/2001/XMLSchema}string	

2.4.3 Assignment and Routing Policy

This section enables you to assign participants to the task and create a policy for routing the task through the workflow.

[Figure 2–17](#) shows the **Assignment and Routing Policy** section of the Human Task Editor.

Figure 2–17 Human Task Editor — Assignment and Routing Policy Section



2.4.4 Expiration and Escalation Policy

This section enables you to specify the expiration duration of a task.

Figure 2–18 shows the **Expiration and Escalation Policy** section of the Human Task Editor.

Figure 2–18 Human Task Editor — Expiration and Escalation Policy Section

The screenshot shows the Expiration and Escalation Policy section of the Human Task Editor. It features a form with the following elements:

- Escalate after:** A dropdown menu.
- Fixed Duration:** A radio button selected, followed by three input fields for Day (0), Hour (0), and Minutes (0).
- By Expression:** An unselected radio button followed by a text input field and a small icon.
- Maximum Escalation Levels:** A text input field containing the value 0.
- Highest Approver Title:** A dropdown menu.
- Use Due Date:** A checked checkbox followed by a dropdown menu set to 'By Duration' and three input fields for Day (0), Hour (0), and Minutes (0).

 The interface includes a title bar with 'Expiration and Escalation Policy'.

2.4.5 Notification Settings

This section enables you to create and send notifications when a user is assigned a task or informed that the status of the task has changed.

Figure 2–19 shows the **Notification Settings** section of the Human Task Editor (when fully expanded).

Figure 2–19 Human Task Editor — Notification Settings Section

Task Status	Recipient	Notification Header
Assign	Assignees	
Complete	Initiator	
Error	Owner	

No reminders

Encoding:

Make notifications secure (exclude details)

Make notification actionable

Send task attachments with email notifications

Notification header attributes

Name: Value

2.4.6 Advanced Settings

This section enables you to specify advanced design features for the Human Task Editor.

[Figure 2–20](#) shows the advanced settings section of the Human Task Editor.

Figure 2–20 Human Task Editor — Advanced Settings Section

Advanced Settings

Specify Escalation Rule.

Specify WordML stylesheet for attachments.

Specify stylesheet for attachments.

Specify multilingual settings.

Specify callback class on task status.

Specify workflow signature policy.

Override default access to task content and actions

Specify Restricted Assignment

Allow task and routing customization in BPEL callbacks

Show Complete Graphical History

2.4.7 Annotations

This section enables you to label different attributes of the task definition. Annotations are used with Oracle Business Process Analysis. Annotations are used to label different attributes of the task definition.

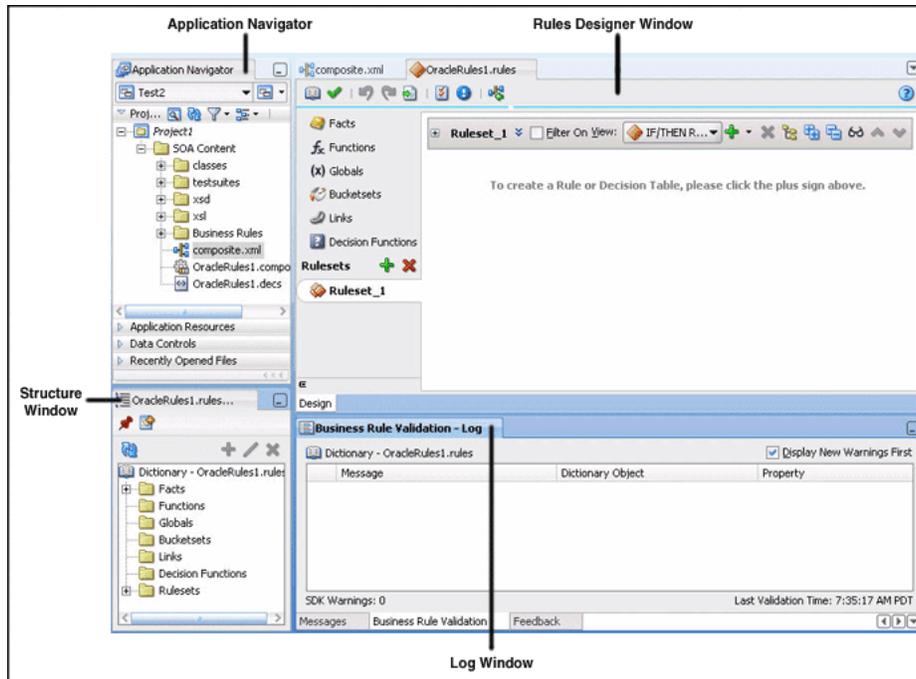
For more information on using the main sections of the Human Task Editor to create workflow tasks, see [Chapter 25, "Designing Human Tasks."](#)

2.5 Introduction to the Business Rules Designer

You can create a business rules service component in the SOA composite application of Oracle JDeveloper and then design it by using the Business Rules Designer, which is displayed when you double-click a business rule in the SOA Composite Editor.

The Business Rules Designer consists of the following main sections shown in [Figure 2–21](#). These sections enable you to work with business rules in Oracle JDeveloper.

Figure 2–21 Rules Designer in Oracle JDeveloper



The main sections of the Business Rules Designer are described in the following list:

2.5.1 Application Navigator

The Application Navigator displays the files in the project. Each project can only contain one composite. But each composite can have multiple components of either the same type or different types (Business Rules, BPEL process, Oracle Mediator, and human workflow).

As you design business rules, additional files, folders, and elements can appear in the Application Navigator.

2.5.2 Rules Designer Window

The **Rules Designer** window provides a visual view of the selected dictionary component. You use the Rules Designer navigation tabs to select different parts of the dictionary that you want to work with. The rules designer window displays when you perform one of the following actions:

- In a composite, double-click a **Business Rule** component.
- Double-click the Business Rule component in the SOA Composite Editor.
- In a BPEL process, double click a business rule.

- In the application navigator, double-click a business rules dictionary file (a file with the **.rules** extension)
- Click the **Design** tab with a **.rules** file selected.

[Table 2–2](#) describes where you can find information about working with a dictionary with Rules Designer.

Table 2–2 Rules Designer Navigation Areas Descriptions

Rules Designer Navigation Tab	Description
Facts	Facts are the objects that rules reason on.
Functions	A function, in Oracle Business Rules, refers to the standard mathematical functions.
Globals	A global, in Oracle Business Rules, is similar to a public static variable in Java.
Bucketsets	Bucketsets define the data types of fact properties.
Links	Links are used to link to a dictionary in the same application or in another application.
Decision Functions	A Decision Function is a function that is configured declaratively, without using RL Language programming.
Rulesets with Rules and Decision Tables	A ruleset provides a unit of execution for rules and for Decision Tables. A Decision Table provides a mechanism for describing data processing tasks.

For more information about the Rules Designer navigation areas and its descriptions, see *Oracle Fusion Middleware User's Guide for Oracle Business Rules*.

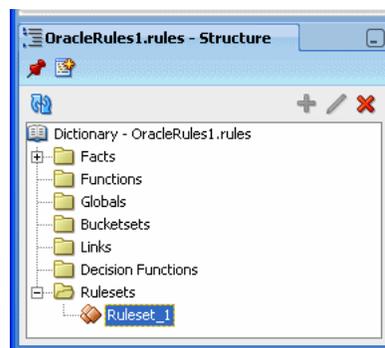
2.5.3 Structure Window

The Structure window offers a structural view of the data in the Business Rule dictionary currently selected in the **Rules Designer** window. You can perform a variety of tasks from this section, by selecting an element and right-clicking on the element, including:

- Managing (creating, editing, refreshing, and deleting) elements such as facts, functions, globals, bucketsets, dictionary links, and decision functions
- Accessing rulesets, rules, and Decision Tables

[Figure 2–22](#) shows the Structure window.

Figure 2–22 Structure Window with Rules Designer Dictionary



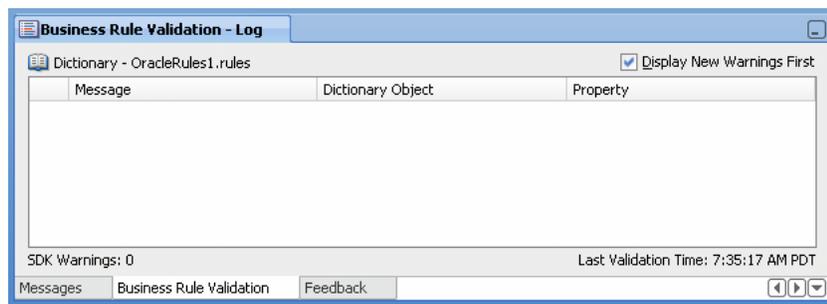
2.5.4 Business Rule Validation Log Window

Rules Designer displays the status of dictionary validation in the business rule validation log, as shown in [Figure 2–23](#).

When a dictionary is invalid, Rules Designer produces a list of warning messages and lists the associated dictionary objects that you can use to locate the dictionary object and to correct the problem. You can safely ignore the validation warnings that you see when you create rules using Rules Designer. The validation warnings are removed as you create the rules, but are shown during the intermediate steps. To test or deploy rules, the associated dictionary must not display warnings.

For more information on business rules validation, see *Oracle Fusion Middleware User's Guide for Oracle Business Rules*.

Figure 2–23 Rules Designer Business Rule Validation Log

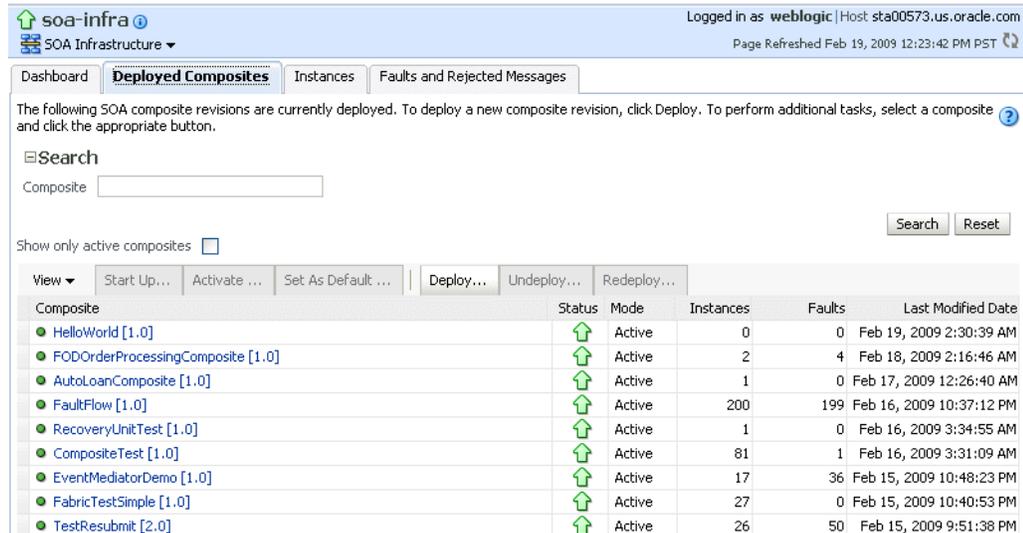


2.6 Introduction to Oracle Enterprise Manager

You can configure, monitor, and manage your SOA composite application during runtime from Oracle Enterprise Manager Fusion Middleware Control Console.

[Figure 2–24](#) shows the Oracle Enterprise Manager Fusion Middleware Control Console with the Deployed Composites tab displayed.

Figure 2–24 Oracle Enterprise Manager Fusion Middleware Control Console



For more information about Oracle Enterprise Manager, see *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite*.

Introduction to the SOA Sample Application

The WebLogic Fusion Order Demo module of the Fusion Order Demo application demonstrates various capabilities of Oracle SOA Suite and is used as an example throughout this guide.

The role of this module is to process the orders of a hypothetical web shopping storefront.

This chapter includes the following sections:

- [Section 3.1, "Introduction to the WebLogic Fusion Order Demo Application"](#)
- [Section 3.2, "Setting Up the WebLogic Fusion Order Demo Application"](#)
- [Section 3.3, "Taking a Look at the WebLogic Fusion Order Demo Application"](#)

3.1 Introduction to the WebLogic Fusion Order Demo Application

The WebLogic Fusion Order Demo application is part of a larger sample application called the Fusion Order Demo application. In this larger sample application, Global Company sells electronic devices through many channels, including a web-based client application. Electronic devices are sold through a storefront-type web application. Customers can visit the web site, register, and place orders for the products.

There are two parts to the Fusion Order Demo: the Store Front module and the WebLogic Fusion Order Demo module.

3.1.1 The Store Front Module

The Store Front module provides a rich UI built with Oracle Application Development Framework to show how to combine an easily built AJAX user interface with a sophisticated SOA composite application. It is based on Oracle ADF business components, ADF model data bindings, and ADF faces.

The Store Front module uses a scenario in which customers can visit a web site to register and place orders for products.

[Figure 3-1](#) shows the Home page of the Store Front module user interface. It shows the featured products that the site wishes to promote and gives access to the full catalog of items. Products are presented as images along with the name of the product. Page regions divide the product catalog area from other features that the site offers.

Figure 3–1 Home Page of the Store Front Module User Interface

For more information about the Store Front module, see *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

3.1.2 The WebLogic Fusion Order Demo Module

This sample application shows how to use the SOA paradigm and Oracle SOA Suite to integrate a number of applications into one cohesive ordering system. The integrated applications can be both internal and external to an enterprise.

Oracle SOA Suite components used by the WebLogic Fusion Order Demo are:

- Oracle Mediator
- Oracle BPEL Process
- Human Task
- Oracle Business Rules
- Oracle User Messaging Service

Samples demonstrating the use of these components can be found at:

http://www.oracle.com/technology/sample_code/products/soa

Once an order has been placed by using the Store Front module, the WebLogic Fusion Order Demo application processes the order. When processing an order, it uses various internal and external applications, including a customer service application, a credit validation system, and both an internal vendor and external vendor. For example, the internal vendor (Warehouse) and external vendor (PartnerSupplier), 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.

As it is being processed by the WebLogic Fusion Order Demo module, the order can be monitored by using the Fusion Middleware Control Console.

For information about SOA composite applications, see [Chapter 1, "Introduction to SOA Composite Applications"](#).

3.2 Setting Up the WebLogic Fusion Order Demo Application

To set up the WebLogic Fusion Order Demo, you need to download the application resources, then install and run the WebLogic Fusion Order Demo module. For specific instructions on setting up your development environment and running the WebLogic

Fusion Order Demo application, see *Oracle Fusion Middleware Tutorial for Running and Building an Application with Oracle SOA Suite*.

Note: You download the application resources to a directory that is referred to in this document as *DEMO_DOWNLOAD_HOME*. When you create the WebLogicFusionOrderDemo application, you will create the application in a working application directory, such as `C:\fod`. You will copy needed files from the *DEMO_DOWNLOAD_HOME* directory to the working application directory.

3.3 Taking a Look at the WebLogic Fusion Order Demo Application

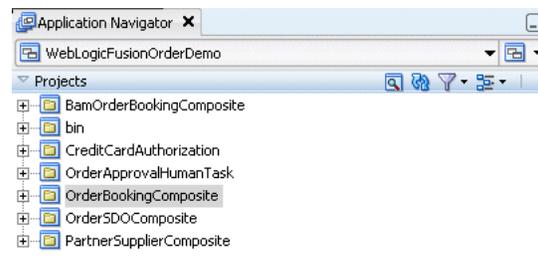
After you have set up the WebLogic Fusion Order Demo application, spend time viewing its artifacts in JDeveloper.

To open the WebLogic Fusion Order Demo in JDeveloper:

1. From the JDeveloper main menu, choose **File > Open**.
2. In the Open dialog, browse to *DEMO_DOWNLOAD_HOME/CompositeServices* and select **WebLogic Fusion Order Demo.jws**. Click **Open**.

Figure 3–2 shows the Application Navigator after you open the file for the application workspace. It displays the project applications of the WebLogic Fusion Order Demo.

Figure 3–2 Projects of WebLogic Fusion Order Demo Application



3.3.1 Project Applications of the WebLogic Fusion Order Demo Application

Table 3–1 lists and describes the projects in the WebLogicFusionOrderDemo application workspace.

Table 3–1 Projects in the WebLogic Fusion Order Demo Application

Application	Description
BamOrderBookingComposite	Contains the OrderBookingComposite composite with Oracle BAM additions. Specifically, it uses the Oracle BAM adapter and Oracle BAM sensors to send active data into Oracle BAM dashboard.
bin	Contains a build script for deploying all the SOA projects.
CreditCardAuthorization	Provides the service needed by OrderBookingComposite project to verify the credit card information of a customer.
OrderApprovalHumanTask	Provides a task form for approving orders.

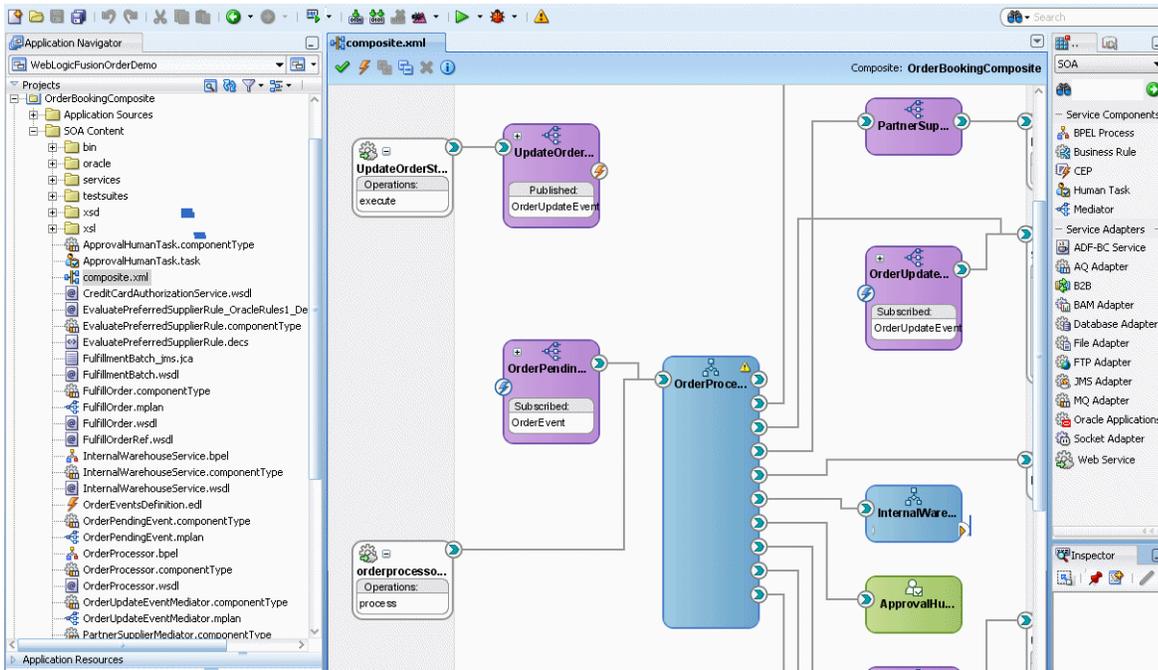
Table 3–1 (Cont.) Projects in the WebLogic Fusion Order Demo Application

Application	Description
OrderBookingComposite	The main project of the WebLogic Fusion Order Demo application described in this guide. It processes an order submitted in the StoreFront service UI of Fusion Order Demo.
OrderSDOComposite	Simulates the StoreFrontService service of the StoreFront application for testing purposes.
PartnerSupplierComposite	Contains a composite containing a BPEL process for obtaining a quote from a partner warehouse. It is referenced as a service from the composite for the OrderBookingComposite project.

3.3.2 The composite.xml File

To understand how a composite is put together, take a look at the main project, namely, `OrderBookingComposite`, in Oracle JDeveloper. To do this, in Application Navigator, expand **OrderBookingComposite > SOA Content** and select **composite.xml**. The composite then appears in the SOA Composite Editor in Oracle JDeveloper, as shown in [Figure 3–3](#).

Figure 3–3 SOA Composite Editor



For details about building and running the WebLogic Fusion Order Demo, see *Oracle Fusion Middleware Tutorial for Running and Building an Application with Oracle SOA Suite*.

Introduction to the Functionality of the SOA Composite Editor

This chapter describes the functionality of the SOA Composite Editor by guiding you through the creation of service components, binding components, and wires in an SOA composite application. This chapter also describes key issues to be aware of when designing your application.

This chapter includes the following sections:

- [Section 4.1, "Introduction to the SOA Composite Editor"](#)
- [Section 4.2, "Designing an SOA Composite Application in Oracle JDeveloper"](#)

4.1 Introduction to the SOA Composite Editor

SOA composite applications consist of the following parts:

- Service binding components
- Composites
- Service components
- Reference binding components
- Wires

For more information about these parts, see [Chapter 1, "Introduction to SOA Composite Applications."](#)

4.2 Designing an SOA Composite Application in Oracle JDeveloper

This section provides an overview of how to create and design an SOA composite application in Oracle JDeveloper. This overview is intended to guide you through the basic steps of component creation, along with describing key issues to be aware of when designing a composite application.

The SOA Composite Editor enables you to use either of two approaches for designing SOA composite applications.

- The top-down approach
- The bottom-up approach

For more information about both approaches, see [Section 1.1, "Introduction to Oracle SOA Suite."](#) The example in this section describes the top-down approach.

For information about designing an end-to-end SOA composite application, see *Oracle Fusion Middleware Tutorial for Running and Building an Application with Oracle SOA Suite*.

WARNING: Always save your changes by selecting **Save All** from the tool bar menu.

4.2.1 How to Create an Application and a Project

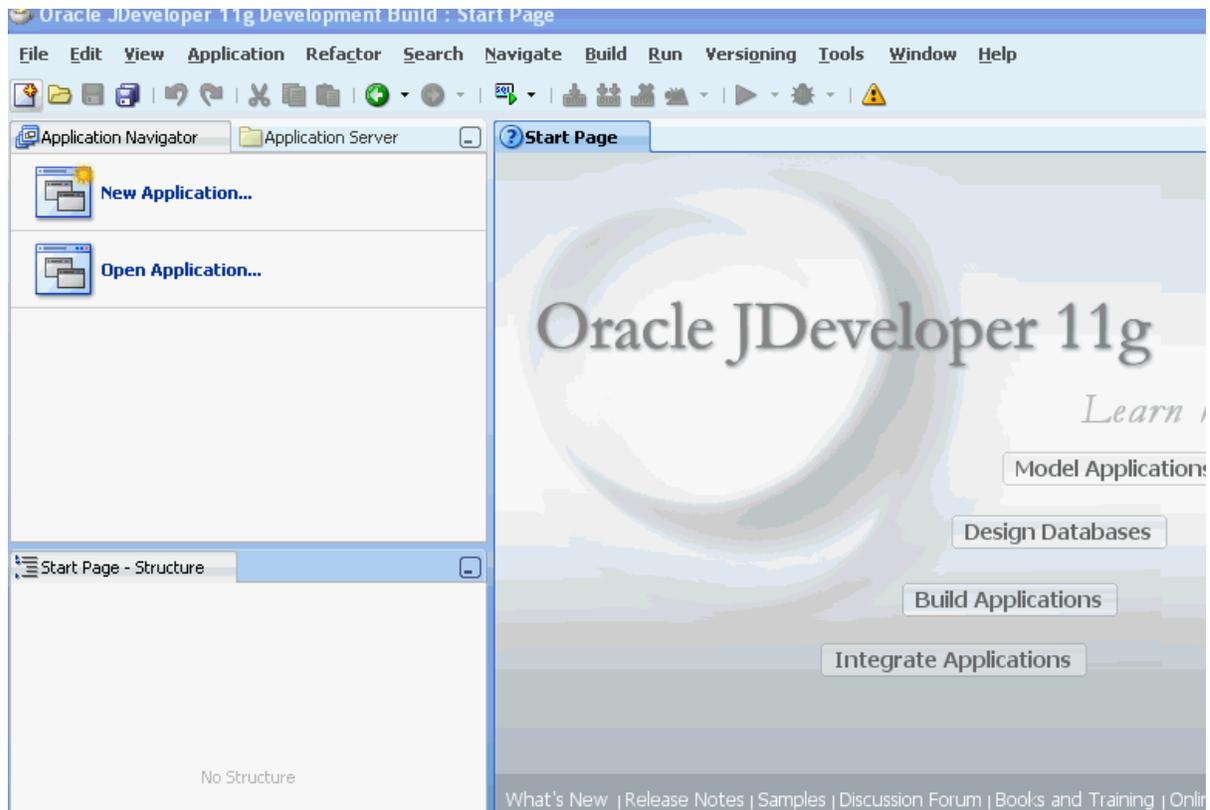
You first create an application for the SOA project.

Note: Oracle SOA Suite is not automatically installed with Oracle JDeveloper. Before you can create an SOA application and project, you must download the SOA Suite extension for Oracle JDeveloper (file name `soa-jdev-extension.zip`) from the Oracle Technology Network and import it into Oracle JDeveloper. For instructions on downloading and installing the SOA Suite extension for Oracle JDeveloper, see *Oracle Fusion Middleware Installation Guide for Oracle JDeveloper*.

To create an application:

1. Start Oracle JDeveloper Studio Edition Version 11.1.1.0.0.
2. If Oracle JDeveloper is running for the first time, specify the location for the Java JDK.

[Figure 4–1](#) shows how Oracle JDeveloper appears the first time you access it.

Figure 4–1 Oracle JDeveloper

3. Create a new SOA composite application, as described in [Table 4–1](#).

Table 4–1 SOA Composite Application Creation

If Oracle JDeveloper...	Then...
Has no applications For example, you are opening Oracle JDeveloper for the first time.	In the Application Navigator in the upper left, click New Application > SOA Application .
Has existing applications	From the File main menu: <ol style="list-style-type: none"> 1. Select New > Applications > SOA Application. 2. Click OK. From the Application menu: <ol style="list-style-type: none"> 1. Select New > Applications > SOA Application. 2. Click OK.

This starts the Create SOA Application wizard.

4. Enter the values shown in [Table 4–2](#):

Table 4–2 SOA Composite Application Creation Values

Field	Value
Application Name	Enter an application name (for this example, MySOAApplication is entered).
Directory Name	Accept the default value or enter a different directory path.

Notes:

- On a UNIX operating system, it is highly recommended that you enable Unicode support by setting the LANG and LC_ALL environment variables to a locale with the UTF-8 character set. This action enables the operating system to process any character in Unicode. SOA technologies are based on Unicode. If the operating system is configured to use non-UTF-8 encoding, SOA components may function in an unexpected way. For example, a non-ASCII file name can make the file inaccessible and cause an error. Oracle does not support problems caused by operating system constraints.

In a design-time environment, if you are using Oracle JDeveloper, select **Tools > Preferences > Environment > Encoding > UTF-8** to enable Unicode support. This is also applicable for runtime environments.

- Do *not* create applications and projects in directory paths that have spaces (for example, `c:\Program Files`).

5. Accept the default values for all remaining settings, and click **Next**.

The Project Name page of the Create SOA Application wizard appears.

6. Enter a name for the project (for this example, `MySOAPProject`), and click **Next**. Note that SOA is automatically selected as the project technology to use.

Note: Composite and component names cannot exceed 500 characters.

A project deployed to the same infrastructure *must* have a unique name across SOA composite applications. This is because the uniqueness of a composite is determined by its project name. For example, do not perform the actions described in [Table 4-3](#). During deployment, the second deployed project (composite) overwrites the first deployed project (composite).

Table 4-3 Restrictions on Naming an SOA Project

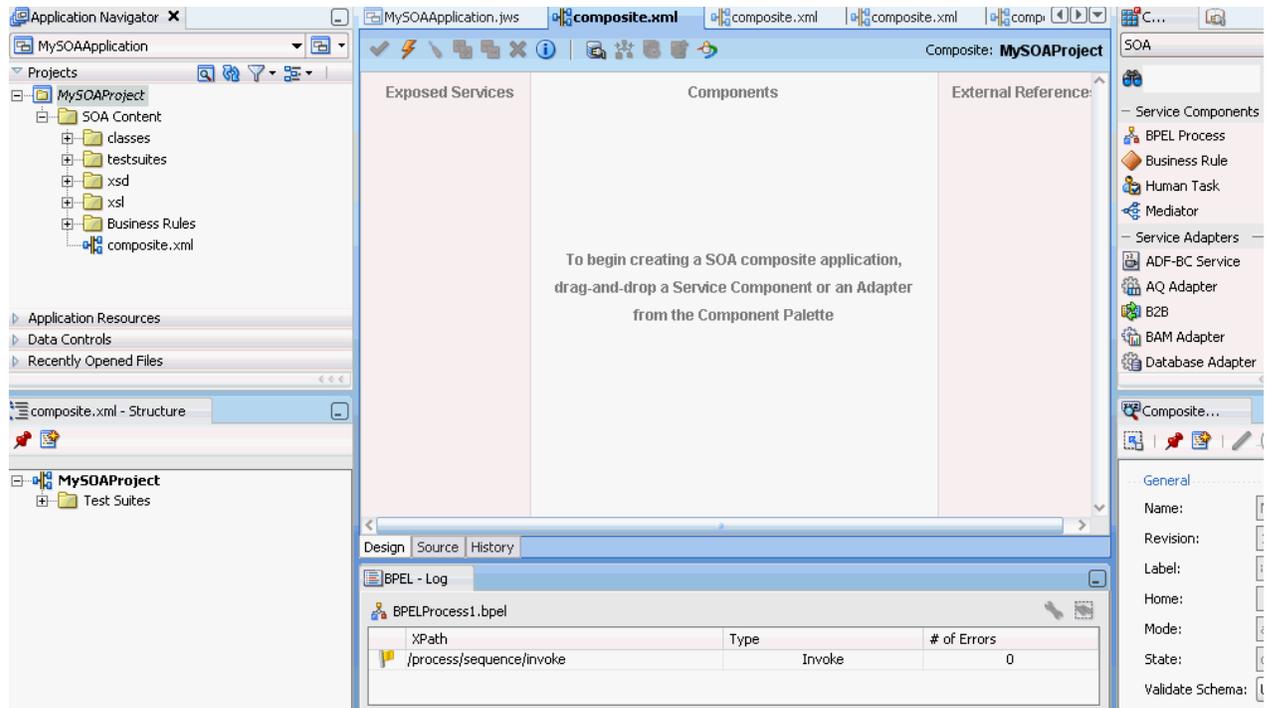
Create an Application Named...	With an SOA Project Named...
Application1	Project1
Application2	Project1

The Project SOA Settings page of the Create SOA Application wizard appears.

7. Select **Empty Composite**, and click **Finish**.

The SOA Composite Editor shown in [Figure 4-2](#) appears. The `composite.xml` file displays in the Application Navigator. This file is automatically created when you create a project. This file describes the entire composite assembly of services, service components, and references. There is one `composite.xml` file per SOA project.

Figure 4–2 SOA Composite Editor



For more information about the sections of the SOA Composite Editor, see [Section 2.1, "Introduction to the SOA Composite Editor."](#)

8. Select **Save All** from the **File** main menu.

4.2.2 How to Add a Service Component

You create service components that implement the business logic or processing rules of your application.

You drag service components into the designer to invoke the initial property editor. This action enables you to define the service interface (and, for asynchronous BPEL processes, an optional callback interface).

[Table 4–4](#) describes the available service components.

Table 4–4 Starting Service Component Editors

Dragging This Service Component...	Invokes The...
BPEL Process	Create BPEL Process dialog: Enables you to create a BPEL process that integrates a series of business activities and services into an end-to-end process flow.
Business Rule	Create Business Rules dialog: Enables you to create a business decision based on rules.
Human Task	Create Human Task dialog: Enables you to create a workflow that describes the tasks for users or groups to perform as part of an end-to-end business process flow.
Mediator	Create Mediator dialog: Enables you to define services that perform message and event routing, filtering, and transformations.

The following example describes the procedures to perform when a BPEL process is dragged into the designer.

To add a service component:

1. From the Component Palette, select **SOA**.
2. From the **Service Components** list, drag a **BPEL Process** into the designer.

The Create BPEL Process dialog appears.

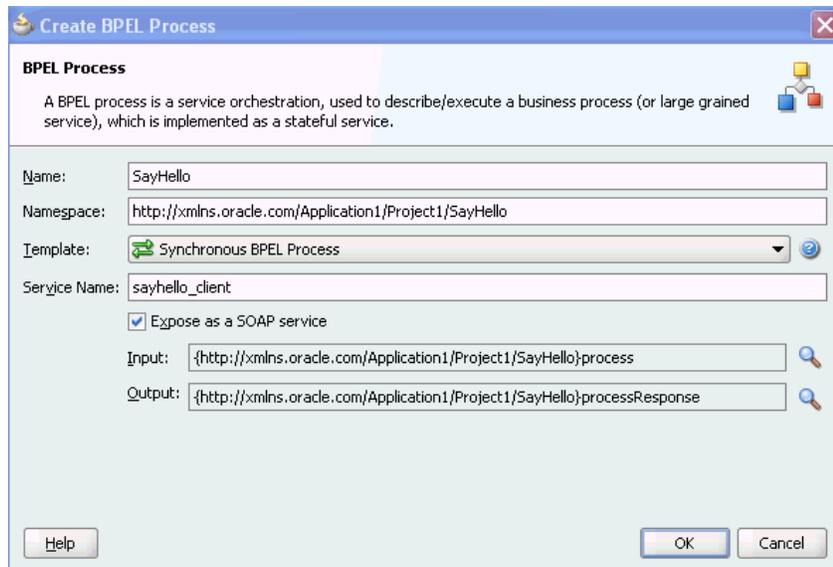
3. Enter the details shown in [Table 4-5](#).

Table 4-5 Create BPEL Process Dialog Fields and Values

Field	Value
Name	Enter a name (for this example, SayHello is entered).
Namespace	Accept the default value.
Template	Select Synchronous BPEL Process . For more information about available templates, see the online help.
Expose as a SOAP Service	Deselect this checkbox. This creates a standalone BPEL process. If you select this checkbox, a BPEL process and inbound web service binding component are each created and connected.

When complete, the Create BPEL Process dialog appears as shown in [Figure 4-3](#).

Figure 4-3 Create BPEL Process Dialog



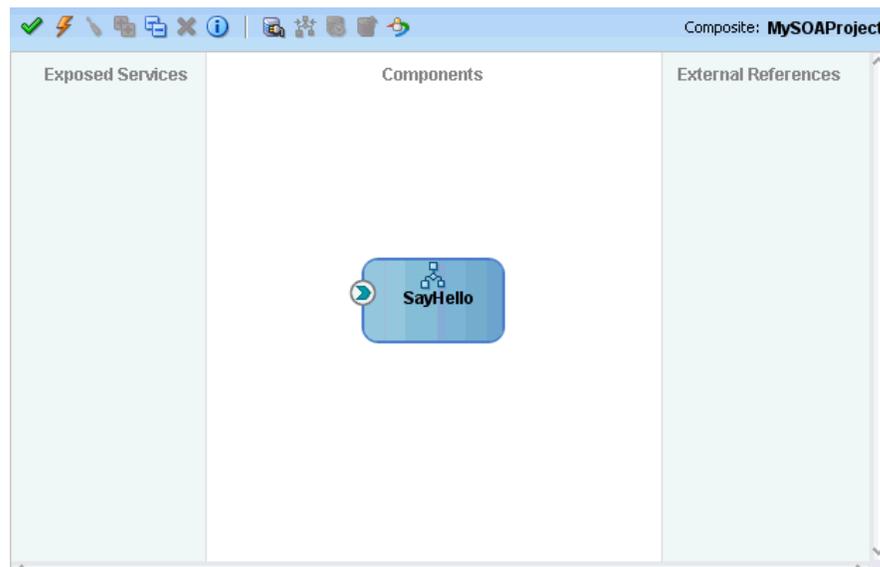
4. Note that the **Input** and **Output** fields also appear in the Create BPEL Process dialog. These fields enable you to select or import specific input and output schemas from the Type Chooser dialog or SOA Resource Browser dialog, respectively. For this example, the default schemas are used, which consist of string input and output values. This schema defines the structure of the message to submit.

The SOA Resource Browser dialog also provides access to the Resource Palette, which provides a single dialog from which to share and access schemas in multiple applications.

5. Accept the default values for all remaining settings.
6. Click **OK**.

The BPEL process displays in the designer shown in [Figure 4-4](#). The single arrow in a circle indicates that this is a synchronous, one-way BPEL process service component. An asynchronous process is indicated by two arrows in a circle, with each pointing in the opposite direction. The two arrows represent an interface and callback interface.

Figure 4-4 BPEL Process



You can more fully define the content of your BPEL process now or at a later time. For this top-down example, the content is defined now.

7. Select **Save All** from the **File** main menu.

4.2.3 What You May Need to Know About Adding and Deleting a Service Component

Note the following details about adding service components:

- A service component can be created from either the SOA Composite Editor or the designer of another component. For example, you can create a human task component from the SOA Composite Editor or the Oracle BPEL Designer.
- The Resource Palette can be used to browse for service components defined in the SOA Composite Editor, and those deployed. A reference and wire are created when a service component from the SOA Composite Editor or from the deployed list is used.

Note the following details about deleting service components:

- You can delete a service component by right-clicking it and selecting **Delete** from the context menu.

- When a service component is deleted, all references pointing to it are invalidated and all wires are removed. The service component is also removed from the Application Navigator.
- A service component created from within another service component can be deleted. For example, a human task created within the BPEL process service component of Oracle JDeveloper can be deleted from the SOA Composite Editor. In addition, the partner link to the task can be deleted. Deleting the partner link removes the reference interface from its `.componentType` file and removes the wire to the task.

4.2.4 How to Edit a Service Component

To define specific details about the service component, you double-click the service component to display the appropriate editor, as described in [Table 4–6](#).

Table 4–6 Starting SOA Service Component Wizards and Dialogs

Double-Clicking This Service Component...	Displays The...
BPEL Process	Oracle BPEL Designer for further designing.
Business Rule	Business Rules Designer for further designing.
Human Task	Human Task Editor for further designing.
Mediator	Oracle Mediator Editor for further designing.

To edit a service component:

1. Double-click the **SayHello** BPEL process.

This opens the BPEL process in editing.

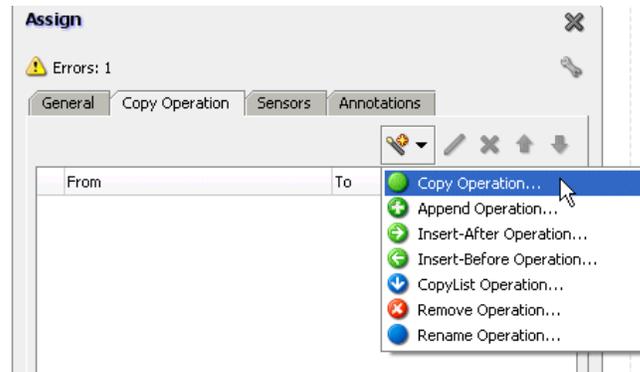
To return to the SOA Composite Editor from within any service component, click **Go to Composite Editor** on the tool bar. You can also double-click **composite.xml** in the Application Navigator or single-click **composite.xml** above the designer.

2. Go to the Component Palette in the upper right, as shown in [Figure 4–5](#).

Figure 4–5 Component Palette



3. Drag an **Assign** activity into the designer beneath the **receiveInput** receive activity.
4. Double-click the **Assign** activity.
5. Click the **Copy Operation** tab.
6. From the dropdown list shown in [Figure 4–6](#), select **Copy Operation**.

Figure 4–6 Copy Operation Selection

7. Enter appropriate details. For this example, the details shown in [Table 4–7](#) are entered.

Table 4–7 Copy Operation Dialog Fields and Values

Field	Value
From	
<ul style="list-style-type: none"> ■ Type Expression ■ Variables <code>concat('Hello',bpws:getVariableData('inputVariable','payload','/client:SayHelloProcessRequest/client:input'))</code> Note: Press Ctrl+Space to access the XPath Expression Builder. Scroll through the list of values that appears and double-click the value you want. As you enter information, a red underscore can appear. This indicates that you are being prompted for additional information. Either enter additional information, or press the Esc key and delete the trailing slash to complete the input of information. 	
To	
<ul style="list-style-type: none"> ■ Type Variable ■ Variables Expand and select Variables > Process > Variables > outputVariable > payload > client:SayHelloProcessResponse > client:result 	

8. Click **OK** to close the Create Copy Operation dialog and the Assign dialog.
9. In the Application Navigator, double-click **composite.xml** in or single-click **composite.xml** above the designer.

This returns you to the SOA Composite Editor.
10. Select **Save All** from the **File** main menu.

4.2.5 How to Add a Service

You add a service binding component to act as the entry point to the SOA composite application from the outside world.

You drag services into the left swim lane to invoke an initial property editor. This action enables you to define the service interface.

[Table 4–8](#) describes the available services.

Table 4–8 Service Editors

Dragging This Service...	Invokes The...
Web service	Create Web Service dialog: Creates a web invocation service.
Adapters	Adapter Configuration Wizard: Guides you through integration of the service with database tables, database queues, file systems, FTP servers, Java Message Services (JMS), IBM WebSphere MQ, BAM servers, sockets, or Oracle E-Business Suite applications.
ADF-BC Service	Create ADF-BC Service dialog: Creates a service data object (SDO) invocation service.
B2B	B2B Wizard: Guides you through selection of a document definition.
EJB Service	Create EJB Service: Creates an Enterprise JavaBeans service for using SDO parameters with Enterprise JavaBeans.

The following example describes the procedures to perform when a web service is dragged into the designer.

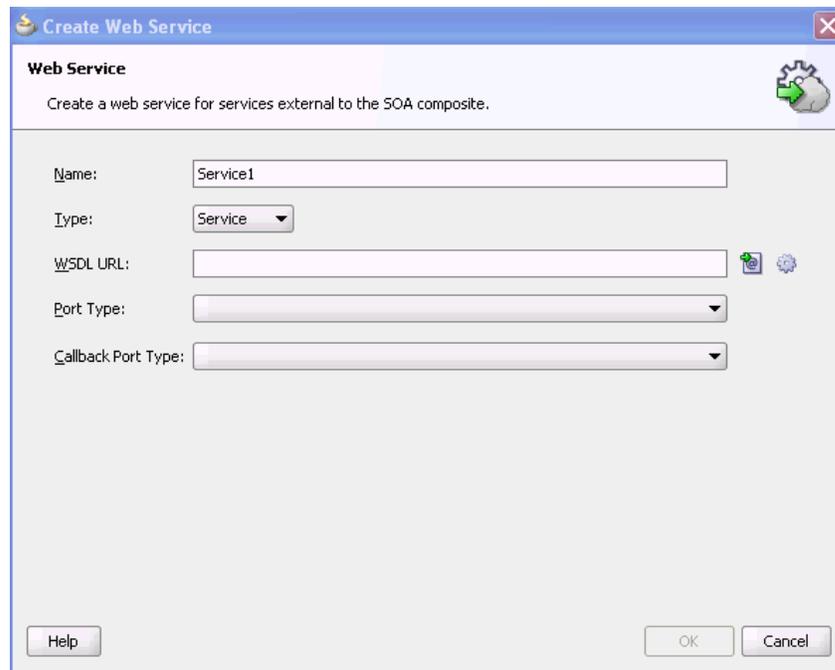
Notes:

- This section describes how to manually create a service binding component. You can also automatically create a service binding component by selecting **Expose as a SOAP Service** when you create a service component. This selection creates an inbound web service binding component that is automatically connected to your BPEL process, human task service, or Oracle Mediator component.
 - You cannot invoke a representational state transfer (REST) service from the SOA Composite Editor.
-
-

To add a service:

1. In the Component Palette, select **SOA**.
2. Drag a **Web Service** to the *left* swim lane.

This invokes the Create Web Service dialog shown in [Figure 4–7](#).

Figure 4–7 Create Web Service Dialog

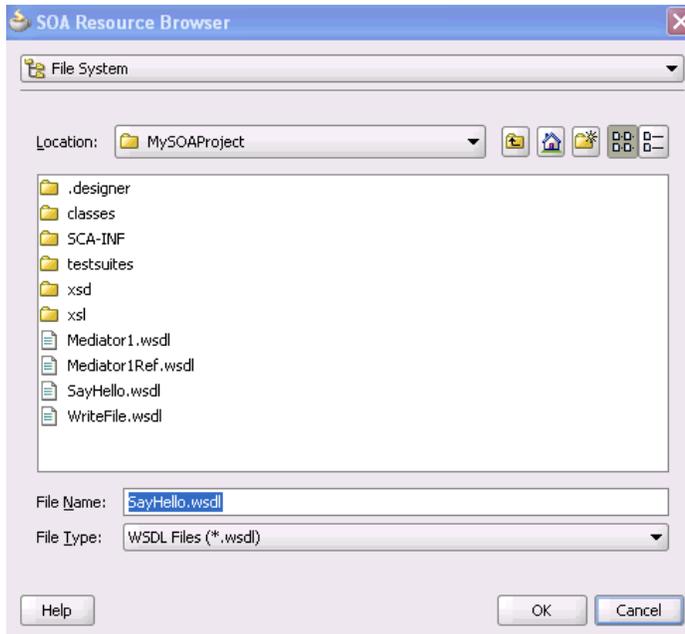
3. Enter the details shown in [Table 4–9](#):

Table 4–9 Create Web Service Dialog Fields and Values

Field	Value
Name	Enter a name for the service (for this example, <i>Service1</i> is entered).
Type	<p>Select the type (message direction) for the web service. Since you dragged the web service to the left swim lane, the Service type is the correct selection, and displays by default:</p> <ul style="list-style-type: none"> ■ Service (default) <ul style="list-style-type: none"> Creates a web service to provide an entry point to the SOA composite application ■ Reference <ul style="list-style-type: none"> Creates a web service to provide access to an external service in the outside world <p>Since this example describes how to create an entry point to the SOA composite application, Service is selected.</p>

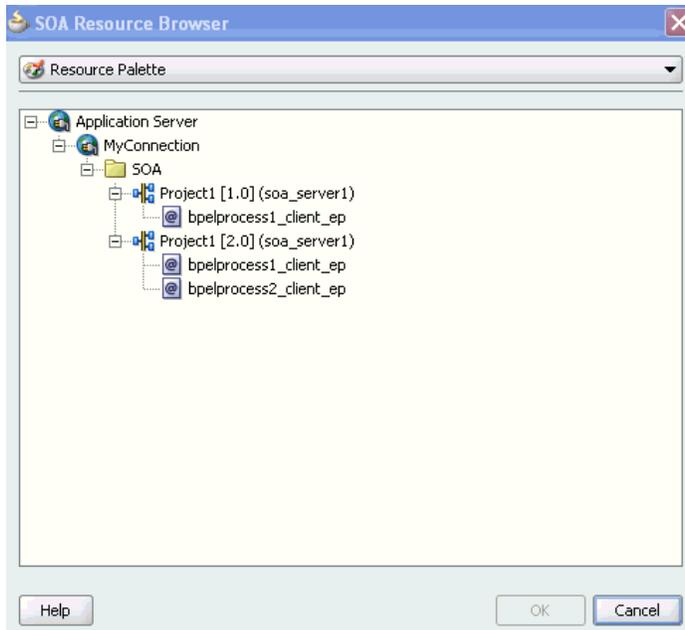
4. Select the WSDL file for the service. There are three methods for selection:
 - a. To the right of the **WSDL URL** field, click the first icon and select an existing WSDL file from the local file system (for this example, **SayHello.wsdl** is selected). Note that **File System** in the list at the top of the dialog is automatically selected. [Figure 4–8](#) provides details.

Figure 4–8 WSDL File Selection



- b. To the right of the **WSDL URL** field, click the first icon and select **Resource Palette** from the list at the top of the dialog, as shown in [Figure 4–9](#). This action enables you to use existing WSDL files from other applications.

Figure 4–9 Use of Existing WSDL files from Other Applications



- c. To the right of the **WSDL URL** field, click the second icon to automatically generate a WSDL file from a schema. [Figure 4–10](#) shows the Create WSDL dialog.

Figure 4–10 Automatic Generation of WSDL File

Specify WSDL message schema(s).

Request Reply Fault Callback

Define Schema for Native Format

URL: xsd/SayHello.xsd

Schema Element: SayHelloProcessRequest

Operation Name: execute

Port Type Name: execute_ptt

Namespace: http://oracle.com/sca/soapservice/MySOAApplication/MySOAProject/Service1

File Name: Service1.wsdl

Help OK Cancel

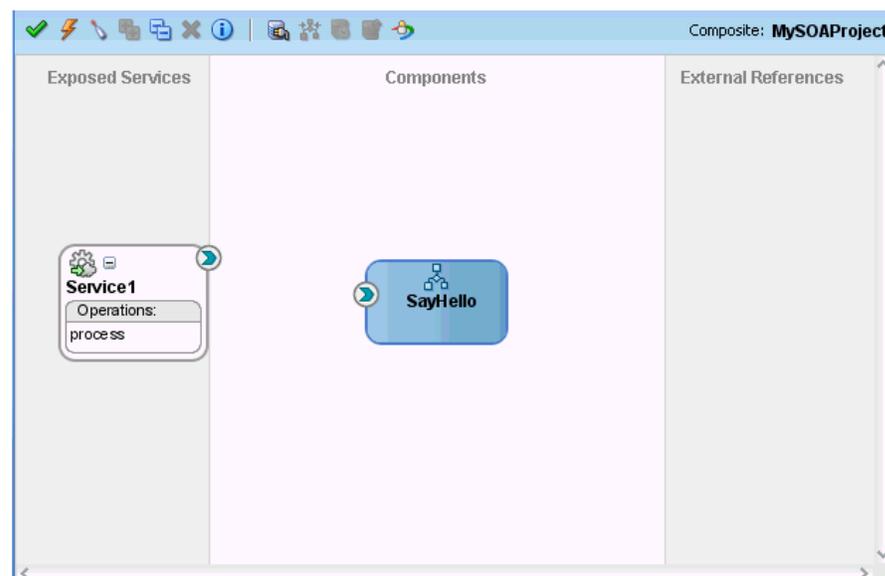
5. Click **OK** to return to the Create Web Service dialog.
6. Note the additional details described in [Table 4–10](#):

Table 4–10 Create Web Service Dialog Fields and Values

Field	Value
Port Type	Displays the port type.
Callback Port Type	Disabled, since this WSDL file is for a synchronous service. This field is enabled for asynchronous services.

7. Click **OK**.

The SOA composite application now looks as shown in [Figure 4–11](#). The service deploys in the left swimlane.

Figure 4–11 SOA Composite Application

8. From the **File** main menu, select **Save All**.

Note: WSDL namespaces must be unique. Do not just copy and rename a WSDL. Ensure that you also change the namespaces.

After initially creating a service, you can edit its contents at a later time. Double-click the component icon to display its appropriate editor or wizard. [Table 4–11](#) provides an overview.

Table 4–11

Double-Click This Service...	To...
Web service	Display the Update Service dialog.
Adapters	Reenter the Adapter Configuration Wizard.
ADF-BC Service	Display the Update Service dialog.
B2B	Reenter the B2B wizard.
EJB Service	Display the Update Service dialog.

4.2.6 What You May Need to Know About Adding and Deleting Services

Note the following detail about adding services:

- When a new service is added for a service component, the service component is notified so that it can make appropriate metadata changes. For example, when a new service is added to a BPEL service component, the BPEL service component is notified to create a partner link that can be connected to a receive or an on-message activity.

Note the following detail about deleting services:

- When a service provided by a service component is deleted, all references to that service component are invalidated and the wires removed.

4.2.7 What You May Need to Know About WSDL References

A WSDL file is added to the SOA composite application whenever you create a new component that has a WSDL (for example, a service binding component, service component (for example, Oracle Mediator, BPEL process, and so on), or reference binding component. When you delete a component, any WSDL imports used by that component are removed *only* if not used by another component. The WSDL import is always removed when the last component that uses it is deleted.

When a service or reference binding component is updated to use a new WSDL, it is handled as if the interface was deleted and a new one was added. Therefore, the old WSDL import is only removed if it is not used by another component.

If a service or reference binding component is updated to use the same WSDL (`porttype QName`), but from a new location, the WSDL import and any other WSDL reference (for example, the BPEL process WSDL that imports an external reference WSDL) are automatically updated to reference the new location.

Simply changing the WSDL location on the source view of the `composite.xml` file's import is not sufficient. Other WSDL references in the metadata are required by the user interface (see the `ui:wSDLLocation` attribute on `composite` and `componentType` services and references). There can also be other WSDL references required by runtime (for example, a WSDL that imports another WSDL, such as the BPEL process WSDL).

Always modify the WSDL location through the dialogs of the SOA Composite Editor in which a WSDL location is specified (for example, a web service, BPEL partner link, and so on). Changing the URL's host address is the exact case in which the SOA Composite Editor automatically updates all WSDL references.

4.2.8 What You May Need to Know About Invoking the Default Revision of a Composite

A WSDL URL that does not contain a revision number is processed by the default composite application. This action enables you to always call the default revision of the called service without having to make other changes in the calling composite.

If you want your WSDL to be processed by the default composite application, but the WSDL URL includes a revision number, you can manually remove it.

Not doing so results in the hard-coded references of the revision number of the service being called in the calling composite. This binds it to always call that particular revision even if the default revision of the called service changes after deployment.

1. In the SOA Composite Editor, double-click the reference binding component that contains the WSDL revision number to remove.

The Update Reference dialog appears.

2. Click the icon to the right of the **WSDL URL** field.

The SOA Resource Browser dialog appears.

3. Select **Resource Palette** from the list at the top of the dialog.
4. Expand the nodes under the appropriate application server connection to list all deployed composites and revisions.
5. Select the appropriate endpoint and click **OK**.

Your selection displays in the **WSDL URL** field.

6. Remove everything between the ! and / symbols. For example, assume the revision number is 3.0. Change:

```
http://pdent12.us.oracle.com:8001/soa-infra/services/default/VersionedComposite!3.0*e295c89a-b198-4835-ab16-a3a250d3b46c/Mediator1_ep?WSDL
```

to:

```
http://pdent12.us.oracle.com:8001/soainfra/services/default/VersionedComposite/Mediator1_ep?WSDL
```

The WSDL reloads.

7. Select port types appropriate to your environment.
8. Click **OK**.

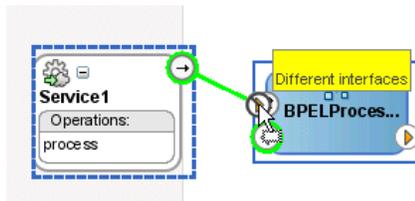
This action enables your WSDL file to be processed by the default composite application.

4.2.9 How to Wire a Service and a Service Component

You wire (connect) the web service and BPEL process service component. Note the following:

- Since the web service is an inbound service, a reference handle displays on the right side. Web services that are outbound references do not have a reference handle on the right side.
- You can drag a defined interface to an undefined interface in either direction (reference to service or service to reference). The undefined interface then inherits the defined interface. There are several exceptions to this rule:
 - A component has the right to reject a new interface. For example, a mediator can only have one inbound service. Therefore, it rejects attempts to create a second service.
 - You cannot drag an outbound service (external reference) to a business rule because business rules do not support references. When dragging a wire, the user interface highlights the interfaces that are valid targets.
- You cannot wire services and composites that have different interfaces. For example, you cannot connect a web service configured with a synchronous WSDL file to an asynchronous BPEL process. [Figure 4–12](#) provides details.

Figure 4–12 Limitations on Wiring Services and Composites with Different Interfaces

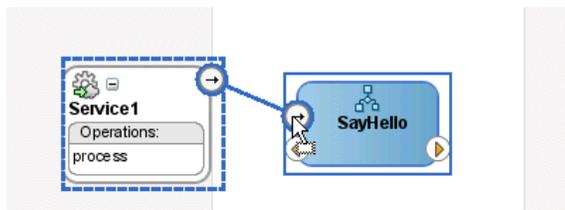


The service and reference must match, meaning the interface and the callback must be the same. If you have two services that have different interfaces, you can place a mediator between the two services and perform a transformation between the interfaces.

To wire a service and a service component:

1. From the **Service1** reference handle, drag a wire to the **SayHello** BPEL process interface, as shown in [Figure 4–13](#).

Figure 4–13 Wire Connection



2. Create additional service components and wire them, as needed.
For more information about `composite.xml` file contents, see [Section 4.2.11, "How to Add a Reference."](#)
3. Select **Save All** from the **File** main menu.

4.2.10 What You May Need to Know About Adding and Deleting Wires

Note the following details about adding wires:

- A service component can be wired to another service component if its reference matches the service of the target service component. Note that the match implies the same interface and callback interface.
- Adding the following wiring between two mediator service components causes an infinite loop:
 - Create a business event.
 - Create a mediator service component and subscribe to the event.
 - Create a second mediator service component to publish the same event.
 - Wire the first mediator to the second mediator component service.

If you remove the wire between the two mediators, then for every message, the second mediator can publish the event once and the first mediator can subscribe to it.

Note the following details about deleting wires:

- When a wire is deleted, the component's outbound reference is automatically deleted and the component is notified so that it can clean up (delete the partner link, clear routing rules, and so on). However, the component's interface is never deleted. All Oracle SOA Suite services are defined by their WSDL interface. When a component's interface is defined, there is no automatic deletion of the service interface in the SOA Composite Editor.

If you want to change the service WSDL interface, there are several workarounds:

- In most cases, you just want to change the schema instead of the inbound service definition. In the SOA Composite Editor, click any interface icon that uses the WSDL. For example, you can click the web service interface icon or the Oracle Mediator service icon. This invokes the Update Interface dialog, which enables you to change the schema for any WSDL message.
- If you are using an Oracle Mediator service component, the **Refresh operations from WSDL** icon of the Oracle Mediator Editor enables you to refresh (after adding new operations) or replace the Oracle Mediator WSDL. However, you are warned if the current operations are to be deleted. If you change the WSDL to the new inbound service WSDL using this icon, the wire typically breaks because the interface has changed. You can then wire Oracle Mediator to the new service.
- In many cases, a new service requires a completely new Oracle Mediator. Delete the old Oracle Mediator, create a new one, and wire it to the new service.
- If you are using a BPEL process service component, select a new WSDL through the Edit Partner Link dialog.

See [Section 4.2.14, "How to Update Message Schemas of Components \(Optional\)"](#) for details about the Update Interface dialog.

4.2.11 How to Add a Reference

You can add reference binding components that enable the SOA composite application to send messages to external services in the outside world.

You drag references into the right swim lane to invoke the initial property editor. This action enables you to define the service interface.

[Table 4–12](#) describes the available references.

Table 4–12 Reference Editors

Dragging This Service...	Invokes The...
Web Service	Create Web Service dialog: Creates a web invocation service.
Adapters	Adapter Configuration Wizard: Guides you through integration of the service with database tables, database queues, file systems, FTP servers, Java Message Services (JMS), IBM WebSphere MQ, BAM servers, sockets, or Oracle E-Business Suite applications.
ADF-BC Service	Create ADF-BC Service dialog: Creates a service data object (SDO) invocation service.
B2B	B2B Wizard: Guides you through selection of a document definition.
EJB Service	Create EJB Service dialog: Creates an Enterprise JavaBeans service for using SDO parameters with Enterprise JavaBeans.

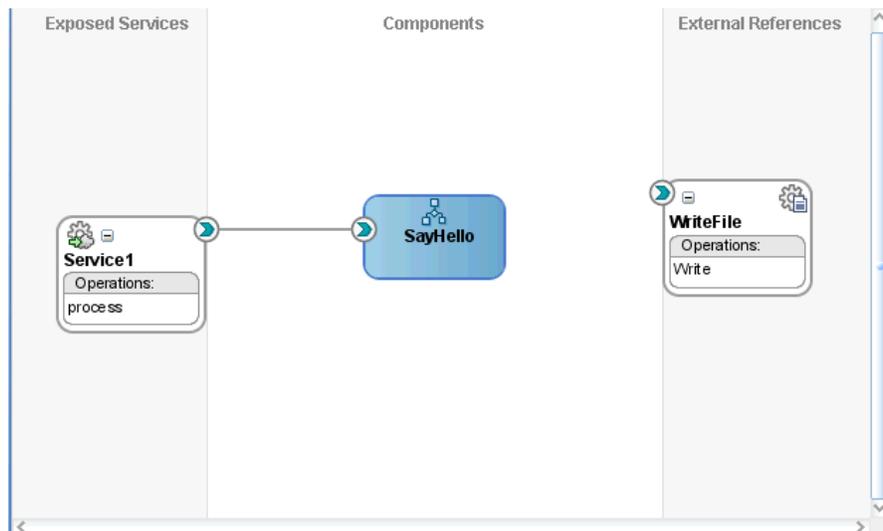
The following example describes the procedures to perform when a file adapter is dragged into the designer.

To add a reference:

1. In the Component Palette, select **SOA**.
2. Drag a **File Adapter** to the *right* swim lane.
This launches the Adapter Configuration wizard.
3. Provide appropriate responses to the dialogs that appear. For example, to configure the file adapter to write to a file in a directory, you are prompted to:
 - Select to write to a file.
 - Specify the directory in which to write the outgoing file.
 - Specify the schema file location.

When complete, the designer looks as shown in [Figure 4–14](#):

Figure 4–14 Design Completion of SOA Composite Application



For more information about how creating partner links within a BPEL process service component impacts how partner links display in the SOA Composite Editor, see [Section 5.4, "Creating a Partner Link."](#)

4. Double-click the **SayHello** BPEL process.
5. Complete the remaining portions of the BPEL process design:
 - Create an invoke activity to invoke the partner link
 - Create a variable
 - Assign a return value to the variable
6. Select **Save All** from the **File** main menu.

After initially creating a reference, you can edit its contents at a later time. Double-click the component icon to display its appropriate editor or wizard. See [Table 4–11](#) for an overview.

4.2.12 What You May Need to Know About Adding and Deleting References

Note the following detail about adding references:

- The only way to add a new reference in the SOA Composite Editor is by wiring the service component to the necessary target service component. When a new reference is added, the service component is notified so it can make appropriate changes to its metadata. For example, when a reference is added to a BPEL service component, the BPEL service component is notified to add a partner link that can then be used in an invoke activity.

Note the following details about deleting references:

- When a reference for a service component is deleted, the associated wire is also deleted and the service component is notified so it can update its metadata. For example, when a reference is deleted from a BPEL service component, the service component is notified to delete the partner link in its BPEL metadata.
- Deleting a reference connected to a wire clears the reference and the wire.

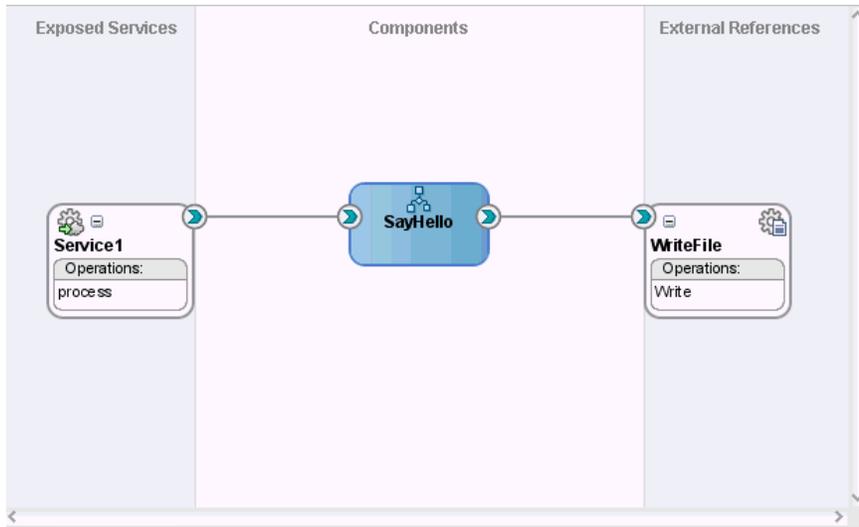
4.2.13 How to Wire a Service Component and a Reference

You now wire (connect) the BPEL process and the file adapter reference.

To wire a service component and a reference:

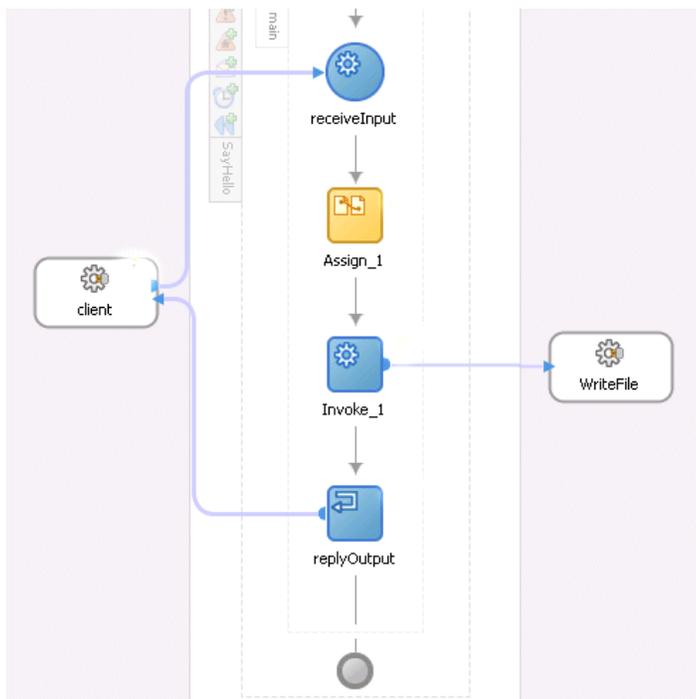
1. In the Application Navigator, double-click **composite.xml** or single-click **composite.xml** above the designer.
2. From the **SayHello** BPEL process, drag a wire to the **WriteFile** reference, as shown in [Figure 4–15](#).

Figure 4–15 Wiring of a Service Component and Reference



3. Double-click the **SayHello** BPEL process and note that the **WriteFile** reference displays as a partner link in the right swim lane, as shown in [Figure 4–16](#).

Figure 4–16 Display of the Reference as a Partner Link in the BPEL Process



4. Select **Save All** from the **File** main menu.
5. In the Application Navigator, select the **composite.xml** file.
6. Click the **Source** tab to review what you have created.

The *Service1* service shown in [Example 4–1](#) provides the entry point to the composite.

Example 4-1 Service

```

<composite name="MySOAProject">
  . . .
  . . .
<service name="Service1" ui:wSDLLocation="SayHello.wsdl">
  <interface.wsdl interface="http://xmlns.oracle.com/MySOAApplication/
    MySOAProject/ SayHello#wsdl.interface(SayHello)"/>
  <binding.ws port="http://xmlns.oracle.com/MySOAApplication/MySOAProject/
    SayHello#wsdl.endpoint
    (sayhello_client_ep/SayHello_pt)"/>
</service>

```

The SayHello BPEL process service component is shown in [Example 4-2](#):

Example 4-2 Service Component

```

<component name="SayHello">
  <implementation.bpel src="SayHello.bpel"/>
</component>

```

A reference binding component named WriteFile is shown in [Example 4-3](#). This reference type is a JCA file adapter. The reference provides access to the external service in the outside world.

Example 4-3 Reference

```

<reference name="WriteFile" ui:wSDLLocation="WriteFile.wsdl">
  <interface.wsdl interface="http://xmlns.oracle.com/pcbpel/adapters/
    file/MySOAApplication/MySOAProject/WriteFile%2F#wsdl.interface(Write_ptt)"/>
  <binding.jca config="WriteFile_file.jca"/>
</reference>

```

In [Example 4-4](#), the communication (or wiring) between service components is described:

- The source Service1 web service is wired to the target SayHello BPEL process service component. Wiring enables web service message communication with this specific BPEL process.
- The source SayHello BPEL process is wired to the target WriteFile reference binding component. This is the reference to the external service in the outside world.

Example 4-4 Wires

```

<wire>
  <source.uri>Service1</source.uri>
  <target.uri>SayHello/sayhello_client</target.uri>
</wire>

<wire>
  <source.uri>SayHello/WriteFile</source.uri>
  <target.uri>WriteFile</target.uri>
</wire>

```

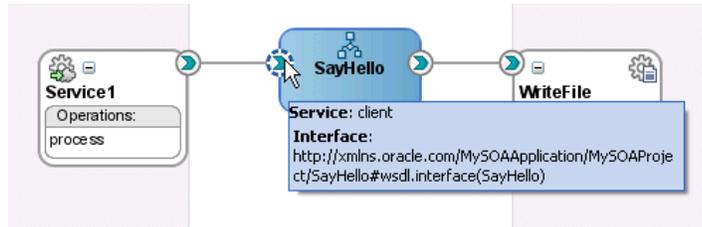
4.2.14 How to Update Message Schemas of Components (Optional)

You can update the message schemas used by service components or binding components.

To update message schemas of components:

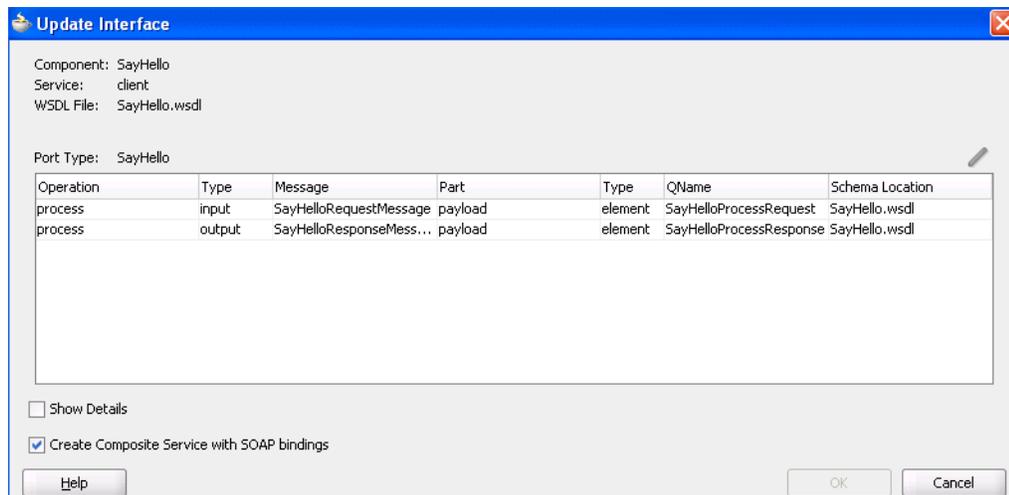
1. Double-click an interface handle of a component, as shown in [Figure 4–17](#). For this example, the inbound interface handle of the **SayHello** BPEL process service component is selected.

Figure 4–17 Selection of Inbound Interface Handle



The Update Interface dialog shown in [Figure 4–18](#) appears. This dialog shows all schemas used by the interface’s WSDL and enables you to choose a new schema for a selected message part.

Figure 4–18 Update Interface Dialog



2. Use one of the following methods to select the message schema to update:
 - Double-click the message schema row.
 - Select a row and click the **Update** icon in the upper right corner above the table.

The Type Chooser dialog appears.

3. Select a new schema element, and click **OK**.
4. In the Update Interface dialog, click **OK**. This updates the interface WSDL to use the new schemas.

4.2.15 What You May Need to Know About Updating Message Schemas of Components

Note the following details about updating message schemas of components:

- It is possible that several operations (or an input and an output) can use the same WSDL message. In this case, the same message is seen in multiple rows of the table. If you update the schema in one row, the change appears in the other rows.
- When the schema used by an interface is changed, it may invalidate previously configured features within a component that depend on the schema. For example, a transformation step in a BPEL process or mediator service component may be invalid because it is using a transformation map created for the old schema.
- If the interface does not have a callback (as is the case for the BPEL process in this example), the Update Interface dialog does show a **Callback Port Type** table.
- Since multiple interfaces can be defined by the same WSDL, the modification to one interaction (WSDL) also modifies the other interfaces.
- When you select **Show Details**, the table shows fully qualified names and complete file paths.
- When the interface belongs to a service component (and not a service binding component or reference), the **Create Composite Service with SOAP bindings** checkbox appears. This checkbox provides the same functionality as the **Expose as a SOAP Service** checkbox on the BPEL process and human task creation dialogs. If you check this box and click **OK**, a service and wire are automatically generated. If it is checked (service exists) and you deselect it and click **OK**, the service and wire are deleted.

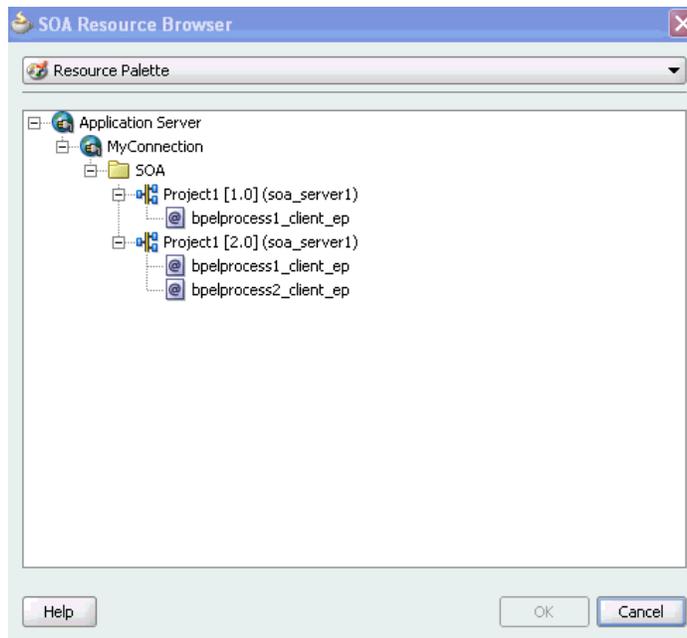
4.2.16 How to Invoke Other Composites

You can invoke other SOA composite applications from your SOA composite application. The other applications must be deployed.

To invoke other composites:

1. Create a web service or partner link through one of the following methods.
 - a. In the SOA Composite Editor, drag a **Web Service** from the Component Palette to the **External References** swim lane.
 - b. In Oracle BPEL Designer, drag a **Partner Link** from the Component Palette to the *right* swim lane.
2. Access the SOA Resource Browser dialog based on the type of service you created.
 - a. For the Create Web Service dialog, click the **Find existing WSDLs** icon.
 - b. For the Edit Partner Link dialog, click the **SOA Resource Browser** icon.
3. From the list at the top, select **Resource Palette**.
4. Expand the tree to display the application server connection to the Oracle WebLogic Administration Server on which the SOA composite application is deployed. For this example, **MyConnection**).
5. Expand the application server connection.
6. Expand the SOA composite applications. [Figure 4-19](#) provides details.

Figure 4–19 Browse for an SOA Composite Application



7. Select the composite service, as shown in [Figure 4–20](#).

Figure 4–20 Selection of Client



8. Click OK.

4.2.17 How to Deploy the SOA Composite Application

Deploying the SOA composite application involves creating and deploying an archive of the SOA composite application. For more information, see [Chapter 43, "Deploying SOA Composite Applications."](#)

4.2.18 How to Manage Deployed Composites

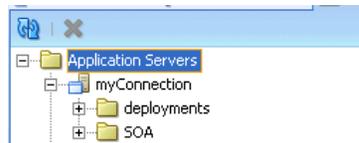
You can manage deployed SOA composite applications from the Application Server Navigator in Oracle JDeveloper. Management tasks consist of deploying, undeploying, activating, retiring, turning on, and turning off SOA composite application revisions.

Note: These instructions assume you have created an application server connection to an Oracle WebLogic Administration Server on which the SOA Infrastructure is deployed. Creating a connection to an Oracle WebLogic Administration Server enables you to browse for managed Oracle WebLogic Servers or clustered Oracle WebLogic Servers in the same domain. From the **File** main menu, select **New > Connections > Application Server Connection** to create a connection.

1. From the **View** main menu, select **Application Server Navigator**.
2. Expand your connection name (for this example, named **myConnection**).

The **deployments** and **SOA** folders appear, as shown in [Figure 4–21](#). The **SOA** folder displays all deployed SOA composite application revisions and services. You can browse all applications deployed on all Oracle WebLogic Administration Servers, managed Oracle WebLogic Servers, and clustered Oracle WebLogic Servers in the same domain. [Figure 4–21](#) provides details.

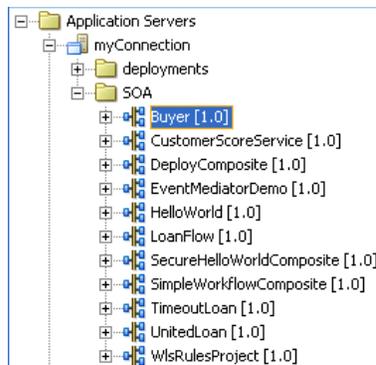
Figure 4–21 Application Server Navigator



3. Expand the **SOA** folder.

Deployed SOA composite applications and services appear, as shown in [Figure 4–22](#).

Figure 4–22 Deployed SOA Composite Applications



4. Right-click a deployed SOA composite application.
5. Select an option to perform. The options that display for selection are based upon the current state of the application. [Table 4–13](#) provides details.

Table 4–13 SOA Composite Application Options

Option	Description
Turned Off	Shuts down a running SOA composite application revision. Any request (initiating or a callback) to the composite is rejected if the composite is shut down. Note: The behavior differs based on which binding component is used. For example, if it is a web service request, it is rejected back to the caller. A JCA adapter binding component may do something else in this case (for example, put the request in a rejected table). This option displays when the composite application has been started.
Turned On	Restarts a composite application revision that was shut down. This action enables new requests to be processed (and not be rejected). No recovery of messages occurs. This option displays when the composite application has been stopped.

Table 4–13 (Cont.) SOA Composite Application Options

Option	Description
Retire	<p>Retires the selected composite revision. If the process life cycle is retired, you cannot create a new instance. Existing instances are allowed to complete normally.</p> <p>An initiating request to the composite application is rejected back to the client. The behavior of different binding components during rejection is equal to that described above for the shut down option.</p> <p>A callback to an initiated composite application instance is delivered properly.</p> <p>This option displays when the composite application is active.</p>
Activate	<p>Activates the retired composite application revision. Note the following behavior with this option:</p> <ul style="list-style-type: none"> ■ All composite applications are automatically active when deployed. ■ Other revisions of a newly deployed composite application remain active (that is, they are not automatically retired). If you want, you must explicitly retire them. <p>This option displays when the application is retired.</p>
Undeploy	<p>Undeploys the selected composite application revision. The consequences of this action are as follows:</p> <ul style="list-style-type: none"> ■ You can no longer configure and monitor this revision of the composite application. ■ You can no longer process instances of this revision of the composite application. ■ You cannot view previously completed processes. ■ The state of currently running instances is changed to stale and no new messages sent to this composite are processed. ■ If you undeploy the default revision of the composite application (for example, 2.0), the next available revision of the composite application becomes the default (for example, 1.0).
Set Default Revision	<p>Sets the selected composite application revision to be the default.</p>

6. If you want to deploy a *prebuilt* SOA composite application archive that includes a deployment profile, right-click the **SOA** folder and select **Deploy SOA Archive**. The archive consists of a JAR file of a single application or an SOA bundle ZIP file containing multiple applications.

You are prompted to select the following:

- The target SOA servers to which you want to deploy the SOA composite application archive.
- The archive to deploy.
- The configuration plan to attach to the application. As you move projects from one environment to another (for example, from testing to production), you typically must modify several environment-specific values, such as JDBC connection strings, hostnames of various servers, and so on. Configuration plans enable you to modify these values using a single text (XML) file called a configuration plan. The configuration plan is created in either Oracle JDeveloper or from the command line. During process deployment, the configuration plan is used to search the SOA project for values that must be replaced to adapt the project to the next target environment. This is an optional selection.

- Whether you want to overwrite an existing composite of the same revision ID. This action enables you to redeploy an application revision.

For more information, see the following documentation:

- [Chapter 43, "Deploying SOA Composite Applications"](#) for details about creating a deployment profile and a configuration plan and deploying an existing SOA archive
- *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite* for details about managing deployed SOA composite applications from Oracle Enterprise Manager Fusion Middleware Control Console

4.2.19 How to Test the SOA Composite Application

You can run and test instances of deployed SOA composite applications from Oracle Enterprise Manager Grid Control Console. For more information about testing an SOA composite application, see *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite*.

Part II

Using the BPEL Process Service Component

This part describes the BPEL process service component.

This part contains the following chapters:

- [Chapter 5, "Getting Started with Oracle BPEL Process Manager"](#)
- [Chapter 6, "Introduction to Interaction Patterns in a BPEL Process"](#)
- [Chapter 7, "Manipulating XML Data in a BPEL Process"](#)
- [Chapter 8, "Invoking a Synchronous Web Service from a BPEL Process"](#)
- [Chapter 9, "Invoking an Asynchronous Web Service from a BPEL Process"](#)
- [Chapter 10, "Using Parallel Flow in a BPEL Process"](#)
- [Chapter 11, "Using Conditional Branching in a BPEL Process"](#)
- [Chapter 12, "Using Fault Handling in a BPEL Process"](#)
- [Chapter 13, "Incorporating Java and Java EE Code in a BPEL Process"](#)
- [Chapter 14, "Using Events and Timeouts in BPEL Processes"](#)
- [Chapter 15, "Coordinating Master and Detail Processes"](#)
- [Chapter 16, "Using the Notification Service"](#)
- [Chapter 17, "Using Oracle BPEL Process Manager Sensors"](#)

Getting Started with Oracle BPEL Process Manager

This chapter describes how to get started with Oracle BPEL Process Manager. Key BPEL design features such as activities, partner links, and adapters are also described.

This chapter includes the following sections:

- [Section 5.1, "Introduction to the BPEL Process Service Component"](#)
- [Section 5.2, "Introduction to Activities"](#)
- [Section 5.3, "Introduction to Partner Links"](#)
- [Section 5.4, "Creating a Partner Link"](#)
- [Section 5.5, "Introduction to Technology Adapters"](#)

5.1 Introduction to the BPEL Process Service Component

This section provides an introduction to the BPEL process service component in the design environment.

5.1.1 How to Add a BPEL Process Service Component

You add BPEL process service components in the SOA Composite Editor.

To create a BPEL process service component:

1. Follow the instructions in [Table 5–1](#) to start Oracle JDeveloper.

Table 5–1 Starting Oracle JDeveloper

To Start...	On Windows...	On UNIX...
Oracle JDeveloper	Click <i>JDev_Oracle_Home\jdeveloper\JDev\bin\jdev.exe</i> or create a shortcut	<code>\$ORACLE_HOME/jdev/bin/jdev</code>

2. Add a BPEL process service component through one of the following methods:

As a service component in an existing SOA composite application:

- a. From the Component Palette, drag a **BPEL Process** service component into the SOA Composite Editor.

In a new application:

- a. From the Application Navigator, select **File > New > Applications > SOA Application**.
- This starts the Create SOA Application wizard.
- b. In the Application Name dialog, enter an application name in the **Application Name** field.
 - c. In the **Directory** field, enter a directory path in which to create the SOA composite application and project.
 - d. Click **Next**.
 - e. In the Project Name dialog, enter a name in the **Project Name** field.
 - f. Click **Next**.
 - g. In the Project SOA Settings dialog, select **Composite with BPEL**.
 - h. Click **Finish**.

Each method causes the Create BPEL Process dialog shown in [Figure 5–1](#) to appear.

Figure 5–1 Create BPEL Process Dialog

3. Provide the required details (including BPEL process name). Click **Help** for details about the types of BPEL processes you can create.

Always use completely unique names when creating BPEL processes. Do not create:

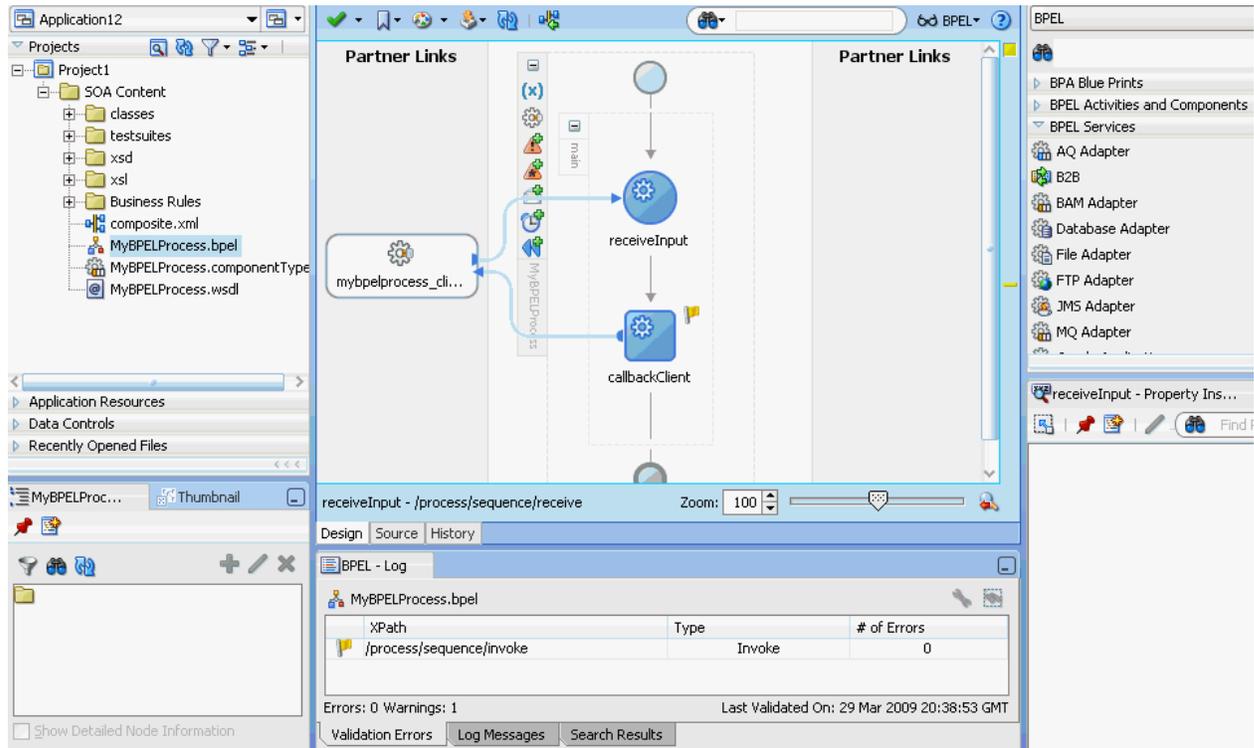
- A process name that begins with a number (for example, 1SayHello)
- A process name that includes a dash (for example, Say-Hello)
- Two processes with the same name, but with different capitalization (for example, SayHello and sayhello)
- A process name that exceeds 500 characters.
- A non-ASCII process name such as that shown in [Figure 5–2](#). The BPEL process name is used in directory and file names of the SOA project, which can cause problems.

Figure 5–2 Non-ASCII BPEL Process Name

BPELProcess100äï

4. Click OK.

Oracle BPEL Designer displays the sections shown in [Figure 5–3](#).

Figure 5–3 Oracle JDeveloper Sections

Each section of Oracle BPEL Designer enables you to perform specific design and deployment tasks.

- Application Navigator
- Design window
- Source window
- History window
- Component Palette
- Property Inspector
- Structure window
- Log window

For a descriptions of these sections, see [Section 2.1, "Introduction to the SOA Composite Editor."](#)

Note: To learn more about these sections, you can also place the cursor in the appropriate section and press **F1** to display online Help.

5.2 Introduction to Activities

Activities are the building blocks of a BPEL process service component. Oracle BPEL Designer includes a set of activities that you drag into a BPEL process service component. You then double-click an activity to define its attributes (property values). Activities enable you to perform specific tasks within a BPEL process service component. For example, here are several key activities:

- An assign activity enables you to manipulate data, such as copying the contents of one variable to another. [Figure 5-4](#) shows an assign activity.

Figure 5-4 Assign Activity



- An invoke activity enables you to invoke a service (identified by its partner link) and specify an operation for this service to perform. [Figure 5-5](#) shows an invoke activity.

Figure 5-5 Invoke Activity

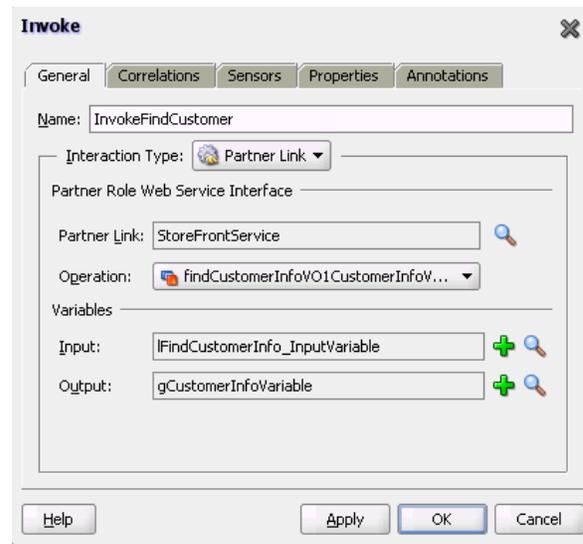


- A receive activity waits for an asynchronous callback response message from a service. [Figure 5-6](#) shows a receive activity.

Figure 5-6 Receive Activity



[Figure 5-7](#) shows an example of a property window (for this example, an invoke activity). In this example, you invoke a partner link named **StoreFrontService** and define its attributes.

Figure 5–7 Invoke Activity Example

The invoke activity enables you to specify an operation you want to invoke for the service (identified by its partner link). The operation can be one-way or request-response on a port provided by the service. You can also automatically create variables in an invoke activity. An invoke activity invokes a synchronous service or initiates an asynchronous web service.

The invoke activity opens a port in the process to send and receive data. It uses this port to submit required data and receive a response. For synchronous callbacks, only one port is needed for both the send and the receive functions.

For more information about activities, see [Chapter A, "BPEL Process Activities and Services"](#) and *Oracle Fusion Middleware Tutorial for Running and Building an Application with Oracle SOA Suite*.

5.3 Introduction to Partner Links

A partner link enables you to define the external services with which the BPEL process service component is to interact. You can define partner links as services or references (for example, through a JCA adapter) in the SOA Composite Editor or within a BPEL process service component in Oracle BPEL Designer. [Figure 5–8](#) shows the partner link icon (in this example, named **WriteRecord**).

Figure 5–8 PartnerLink Icon

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 conversation.

[Figure 5–9](#) shows an example of the attributes of a partner link for a service.

Figure 5–9 Partner Link Dialog

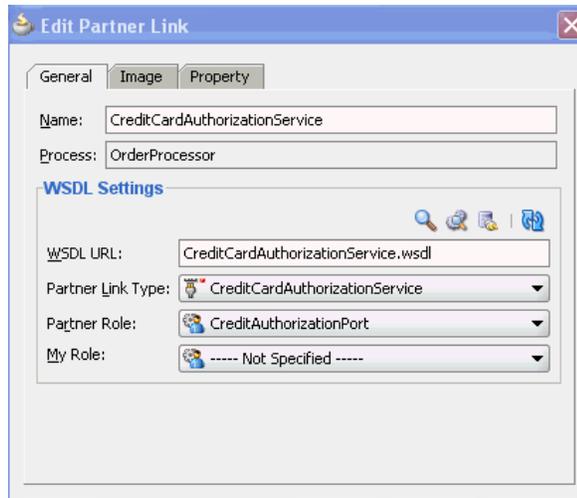


Table 5–2 describes the fields of this dialog.

Table 5–2 Create Partner Link Dialog Fields

Field	Description
Name	A unique and recognizable name you provide for the partner link.
Process	Displays the BPEL process service component name.
WSDL URL	The name and location of the Web Services Description Language (WSDL) file that you select for the partner link. Click the SOA Service Explorer icon (second icon from the left above the WSDL URL field) to access a window for selecting the WSDL file to use.
Partner Link Type	The partner link defined in the WSDL file.
Partner Role	The role performed by the partner link.
My Role	The role performed by the BPEL process service component. In this case, the BPEL process service component does not have a role because it is a synchronous process.

Note: The **Partner Link Type**, **Partner Role**, and **My Role** fields in the Create Partner Link dialog are defined and required by the BPEL standard.

5.4 Creating a Partner Link

The method by which you create partner links within the BPEL process in Oracle BPEL Designer impacts how the partner link displays above in the SOA Composite Editor. This section describes this impact. The WSDL file can be on the local operating system or hosted remotely (in which case you need a URL for the WSDL).

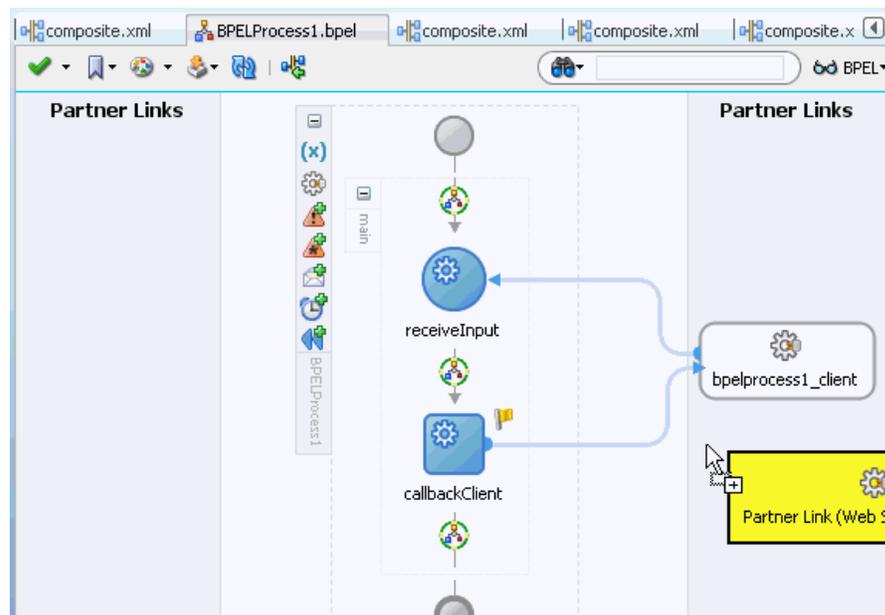
Likewise, creating and wiring a service or reference binding component to a BPEL process service component in the SOA Composite Editor causes a partner link to display in Oracle BPEL Designer.

5.4.1 How to Create a Partner Link

To create a partner link:

1. In the SOA Composite Editor, double-click the BPEL process service component. Oracle BPEL Designer is displayed.
2. In the **Component Palette**, expand **BPEL Services**.
3. Drag a **Partner Link** into the appropriate **Partner Links** swim lane, as shown in [Figure 5–10](#).

Figure 5–10 Partner Link Creation in Oracle BPEL Designer



The Create Partner Link dialog appears.

4. Complete the fields for this dialog, as described in [Table 5–2](#).

For more information about creating a partner link in the SOA Composite Editor, see [Chapter 4, "Introduction to the Functionality of the SOA Composite Editor."](#)

The following sections describe the impact of partner link creation on the SOA Composite Editor.

5.4.1.1 Partner Links for an Outbound Adapter

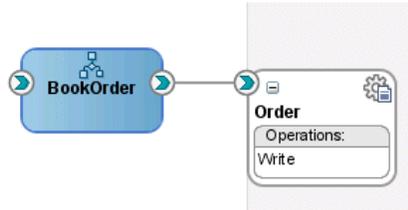
[Table 5–3](#) describes the impact on the SOA Composite Editor.

Table 5–3 Impact of Partner Link Creation on the SOA Composite Editor

Creating the Following for a BPEL Process in Oracle BPEL Designer...	Displays the Following in the SOA Composite Editor...
A partner link for an <i>outbound</i> adapter	<ul style="list-style-type: none"> ■ A reference handle for the BPEL service component ■ A reference representing the outbound adapter in the composite ■ A wire connecting the BPEL service component to the adapter reference

Figure 5–11 shows how this method of creation appears in the SOA Composite Editor.

Figure 5–11 SOA Composite Editor Impact



5.4.1.2 Partner Links for an Inbound Adapter

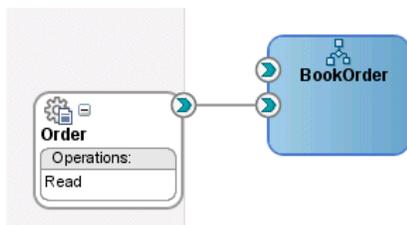
Table 5–4 describes the impact on the SOA Composite Editor.

Table 5–4 Impact of Partner Link Creation on the SOA Composite Editor

Creating the Following for a BPEL Process in Oracle BPEL Designer...	Displays the Following in the SOA Composite Editor...
A partner link for an <i>inbound</i> adapter	<ul style="list-style-type: none"> ■ A service for the BPEL service component ■ A service representing the inbound adapter in the composite ■ A wire connecting the inbound adapter service to the BPEL service component

Figure 5–12 shows how this method of creation appears in the SOA Composite Editor.

Figure 5–12 SOA Composite Editor Impact



5.4.1.3 Partner Links from an Abstract WSDL to Call a Service

Table 5–5 describes the impact on the SOA Composite Editor.

Table 5–5 Impact of Partner Link Creation on the SOA Composite Editor

Creating the Following for a BPEL Process in Oracle BPEL Designer...	Displays the Following in the SOA Composite Editor...
A partner link from an abstract WSDL to call a service	A reference handle with an interface and callback interface defined for the BPEL service component

5.4.1.4 Partner Links from an Abstract WSDL to Implement a Service

Table 5–6 describes the impact on the SOA Composite Editor.

Table 5–6 Impact of Partner Link Creation on the SOA Composite Editor

Creating the Following for a BPEL Process in Oracle BPEL Designer...	Displays the Following in the SOA Composite Editor...
A partner link is created from an abstract WSDL to implement a service	A service with an interface and callback interface for the BPEL service component is created. Note: If an external Simple Object Access Protocol (SOAP) reference with the specified interface and callback interface exists in the SOA Composite Editor, you can either create a new external SOAP reference and wire to it or wire to the existing external SOAP reference.

Figure 5–13 shows how this method of creation appears in the SOA Composite Editor.

Figure 5–13 SOA Composite Editor Impact

5.4.1.5 Partner Links and Human Tasks or Business Rules

Table 5–7 describes the impact on the SOA Composite Editor.

Table 5–7 Impact of Partner Link Creation on the SOA Composite Editor

Creating the Following for a BPEL Process in Oracle BPEL Designer...	Displays the Following in the SOA Composite Editor...
A human task or business rule is created	<ul style="list-style-type: none"> ■ A human task or business rule in the composite ■ A reference for the BPEL service component ■ A wire connecting the BPEL service component to the new human task or decision service

Figure 5–14 shows how this method of creation appears in the SOA Composite Editor.

Figure 5–14 SOA Composite Editor Impact

5.4.1.6 Partner Links from an Existing Human Task, Business Rule, or Oracle Mediator

Table 5–8 describes the impact on the SOA Composite Editor.

Table 5–8 Impact of Partner Link Creation on the SOA Composite Editor

Creating the Following for a BPEL Process in Oracle BPEL Designer...	Displays the Following in the SOA Composite Editor...
A partner link by dragging an existing human task, business rule, or mediator service component from the Resource Palette to the BPEL process	<ul style="list-style-type: none"> ■ A reference for the BPEL service component ■ A wire connecting the BPEL service component to the existing human task, business rule, or mediator

Figure 5–15 shows how this method of creation appears in the SOA Composite Editor.

Figure 5–15 SOA Composite Editor Impact

5.5 Introduction to Technology Adapters

The Partner Link dialog shown in Figure 5–9 also enables you to take advantage of another key feature that Oracle BPEL Process Manager and Oracle JDeveloper provide. Click the **Define Service** icon shown in Figure 5–16 to access the Adapter Configuration wizard.

Figure 5–16 Defining an Adapter

Adapters enable you to integrate the BPEL process service component (and, therefore, the SOA composite application as a whole) with access to file systems, FTP servers, database tables, database queues, sockets, Java Message Services (JMS), MQ, and Oracle E-Business Suite. This wizard enables you to configure the types of adapters shown in Figure 5–17 for use with the BPEL process service component:

Figure 5–17 Adapter Types

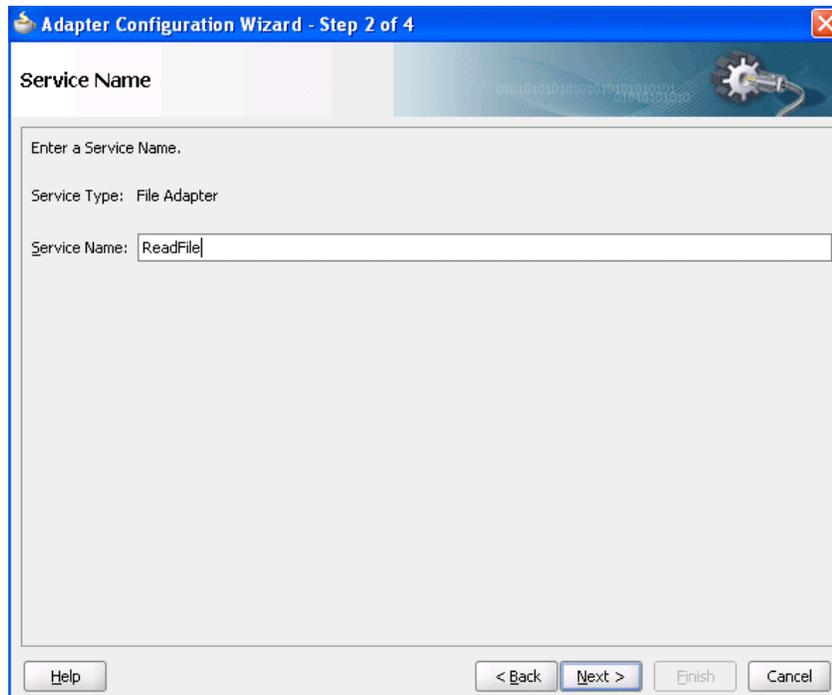
The following adapter types are available:

- **Advanced Queuing (AQ)**
For interaction with a queue. AQ provides a flexible mechanism for bidirectional, asynchronous communication between participating applications.
- **Oracle Business Activity Monitoring (BAM)**
For publishing data to data objects in an Oracle BAM Server.
- **Database**
For interaction with Oracle and non-Oracle databases through JDBC and Oracle Business Intelligence (which is a special data source type).
- **FTP and File**
For file exchange (read and write) on local file systems and remote file systems (through use of the file transfer protocol (FTP)).

Note: When calling the file adapter, Oracle BPEL Process Manager may process the same file twice when run against Oracle Real Application Clusters planned outages. This is because a file adapter is a non-XA compliant adapter. Therefore, when it participates in a global transaction, it may not follow the XA interface specification of processing each file once and only once.

- **Java Messaging Service (JMS)**
For interaction with JMS. The JMS architecture uses a one client interface to many messaging servers architecture.
- **Message Queue (MQ)**
For message exchange with WebSphere MQ queuing systems.
- **Oracle Applications**
For interaction with Oracle Application's set of integrated business applications.
- **Oracle B2B**
- For browsing B2B metadata in the metadata service (MDS) repository and selecting document definitions.
- **Sockets**
For modeling standard or nonstandard protocols for communication over TCP/IP sockets.

When you select an adapter type, the Service Name window shown in [Figure 5-18](#) prompts you to enter a name. For this example, **File Adapter** was selected in [Figure 5-17](#). When the wizard completes, a WSDL file by this service name appears in the Application Navigator for the BPEL process service component (for this example, named **ReadFile.wsdl**). The service name must be unique within the project. This file includes the adapter configuration settings you specify with this wizard. Other configuration files (such as header files and files specific to the adapter) are also created and display in the Application Navigator.

Figure 5–18 Adapter Service Name

The Adapter Configuration wizard windows that appear after the Service Name window are based on the adapter type you selected.

You can also add adapters to your SOA composite application as services or references in the SOA Composite Editor.

For more information about adding adapters to SOA composite applications, see [Chapter 4, "Introduction to the Functionality of the SOA Composite Editor."](#)

For more information about technology adapters, see *Oracle Fusion Middleware User's Guide for Technology Adapters*.

Introduction to Interaction Patterns in a BPEL Process

This chapter describes common interaction patterns between a BPEL process service component and an external service, and shows the best use practices for each.

This chapter includes the following sections:

- Section 6.1, "Introduction to One-Way Messages"
- Section 6.2, "Introduction to Synchronous Interactions"
- Section 6.3, "Introduction to Asynchronous Interactions"
- Section 6.4, "Introduction to Asynchronous Interactions with a Timeout"
- Section 6.5, "Introduction to Asynchronous Interactions with a Notification Timer"
- Section 6.6, "Introduction to One Request, Multiple Responses"
- Section 6.7, "Introduction to One Request, One of Two Possible Responses"
- Section 6.8, "Introduction to One Request, a Mandatory Response, and an Optional Response"
- Section 6.9, "Introduction to Partial Processing"
- Section 6.10, "Introduction to Multiple Application Interactions"

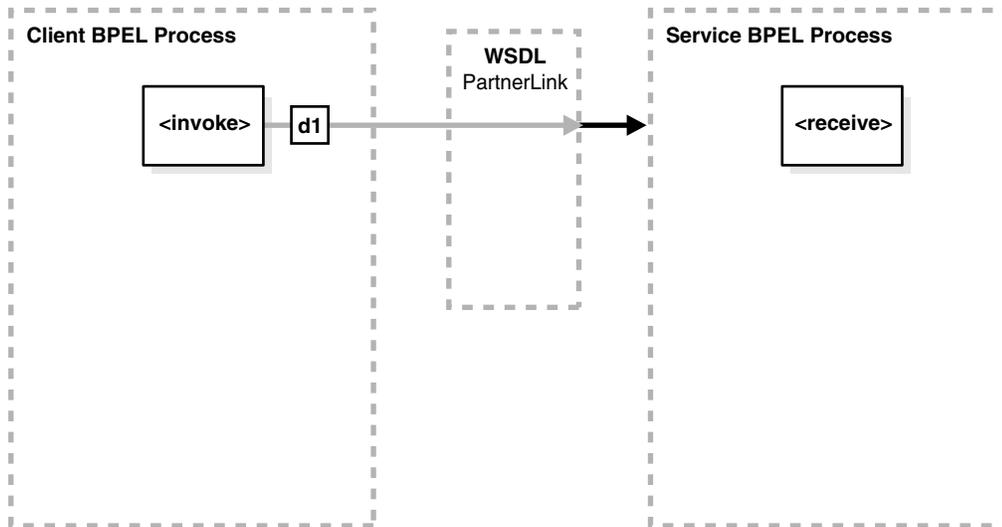
6.1 Introduction to One-Way Messages

In a one-way message, or fire and forget, the client sends a message to the service (d1 in [Figure 6-1](#)), and the service does not need to reply. The client sending the message does not wait for a response, but continues executing immediately. [Example 6-1](#) shows the `portType` and `operation` part of the BPEL process WSDL file for this environment.

Example 6-1 One-Way WSDL File

```
. . .
<wsdl:portType name="BPELProcess1">
  <wsdl:operation name="process">
    <wsdl:input message="client:BPELProcess1RequestMessage" />
  </wsdl:operation>
</wsdl:portType>
. . .
```

[Figure 6-1](#) provides an overview.

Figure 6–1 One-Way Message**BPEL Process Service Component as the Client**

As the client, the BPEL process service component needs a valid partner link and an invoke activity with the target service and the message. As with all partner activities, the Web Services Description Language (WSDL) file defines the interaction.

BPEL Process Service Component as the Service

To accept a message from the client, the BPEL process service component needs a receive activity.

6.2 Introduction to Synchronous Interactions

In a synchronous interaction, a client sends a request to a service (d1 in [Figure 6–2](#)), and receives an immediate reply (d2 in [Figure 6–2](#)). A BPEL process service component can be at either end of this interaction, and must be coded based on its role as either the client or the service. For example, a user requests a subscription to an online newspaper and immediately receives email confirmation that their request has been accepted. [Example 6–2](#) shows the `portType` and `operation` part of the BPEL process WSDL file for this environment.

Example 6–2 Synchronous WSDL File

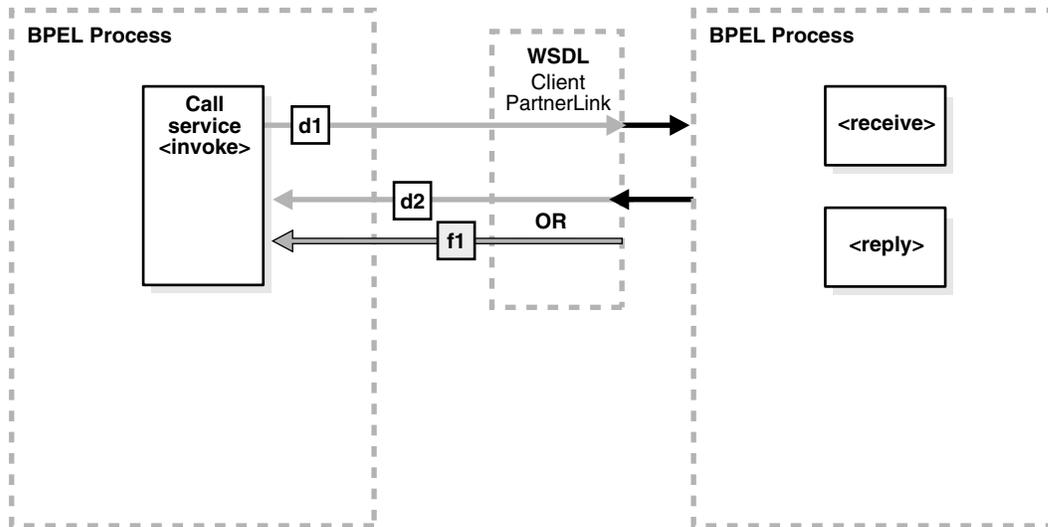
```

. . .
<wsdl:portType name="BPELProcess1">
  <wsdl:operation name="process">
    <wsdl:input message="client:BPELProcess1RequestMessage" />
    <wsdl:output message="client:BPELProcess1ResponseMessage" />
  </wsdl:operation>
</wsdl:portType>

```

[Figure 6–2](#) provides an overview.

Figure 6–2 Synchronous Interaction



BPEL Process Service Component as the Client

When the BPEL process service component is on the client side of a synchronous transaction, it needs an invoke activity. The port on the client side both sends the request and receives the reply. As with all partner activities, the WSDL file defines the interaction.

BPEL Process Service Component as the Service

When the BPEL process service component is on the service side of a synchronous transaction, it needs a receive activity to accept the incoming request, and a reply activity to return either the requested information or an error message (a fault; f1 in Figure 6–2) defined in the WSDL.

For more information about synchronous interactions, see [Chapter 8, "Invoking a Synchronous Web Service from a BPEL Process."](#)

6.3 Introduction to Asynchronous Interactions

In an asynchronous interaction, a client sends a request to a service and waits until the service replies. [Example 6–3](#) shows the `portType` and `operation` part of the BPEL process WSDL file for this environment.

Example 6–3 Asynchronous WSDL File

```

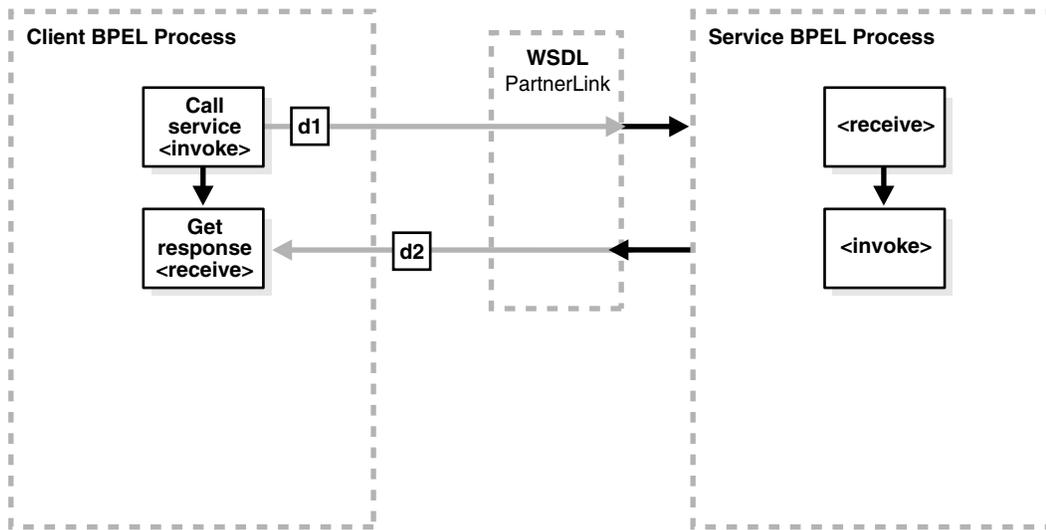
. . .
<wsdl:portType name="BPELProcess1">
  <wsdl:operation name="process">
    <wsdl:input message="client:BPELProcess1RequestMessage" />
  </wsdl:operation>
</wsdl:portType>

. . .
<wsdl:portType name="BPELProcess1Callback">
  <wsdl:operation name="processResponse">
    <wsdl:input message="client:BPELProcess1ResponseMessage" />
  </wsdl:operation>
</wsdl:portType>

```

Figure 6–3 provides an overview.

Figure 6–3 Asynchronous Interaction



BPEL Process Service Component as the Client

When the BPEL process service component is on the client side of an asynchronous transaction, it needs an invoke activity to send the request and a receive activity to receive the reply. As with all partner activities, the WSDL file defines the interaction.

BPEL Process Service Component as the Service

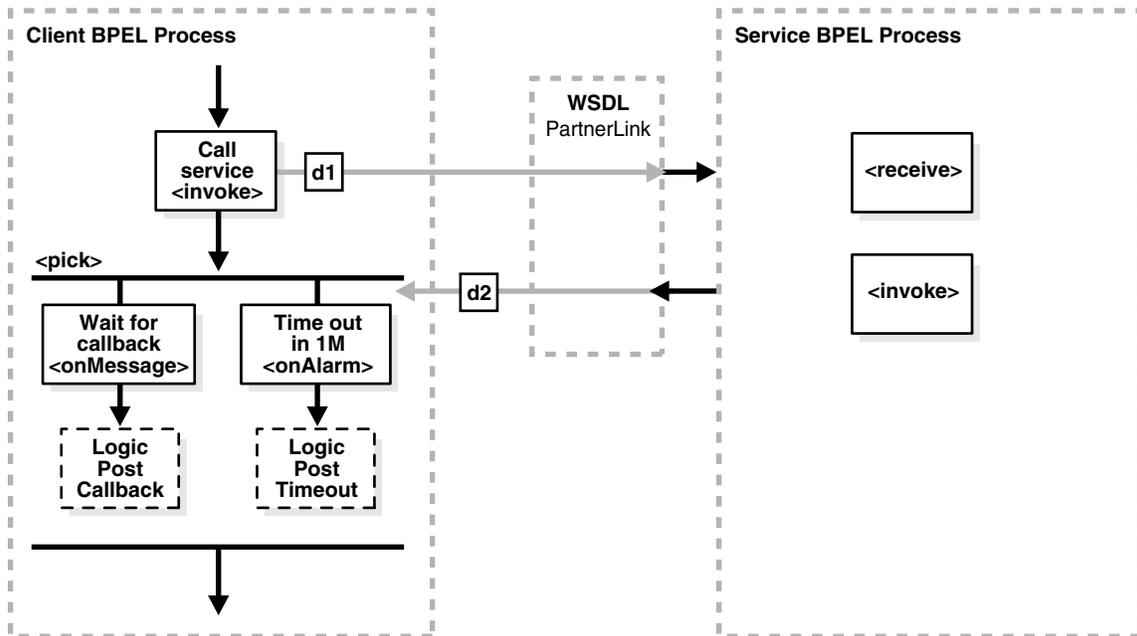
As with a synchronous transaction, when the BPEL process service component is on the service side of an asynchronous transaction, it needs a receive activity to accept the incoming request and an invoke activity to return either the requested information or a fault. Note the difference between this and responding from a synchronous BPEL process: a synchronous BPEL process uses a reply activity to respond to the client and an asynchronous service uses an invoke activity.

For more information about asynchronous interactions, see [Chapter 9, "Invoking an Asynchronous Web Service from a BPEL Process."](#)

6.4 Introduction to Asynchronous Interactions with a Timeout

In an asynchronous interaction with a timeout (which you perform in BPEL with a pick activity), a client sends a request to a service and waits until it receives a reply, or until a certain time limit is reached, whichever comes first. For example, a client requests a loan offer. If the client does not receive a loan offer reply within a specified amount of time, the request is canceled. [Figure 6–4](#) provides an overview.

Figure 6–4 Asynchronous Interaction with Timeout



BPEL Process Service Component as the Client

When the BPEL process service component is on the client side of an asynchronous transaction with a timeout, it needs an invoke activity to send the request and a pick activity with two branches: an onMessage branch and an onAlarm branch. If the reply comes after the time limit has expired, the message goes to the dead letter queue. As with all partner activities, the WSDL file defines the interaction.

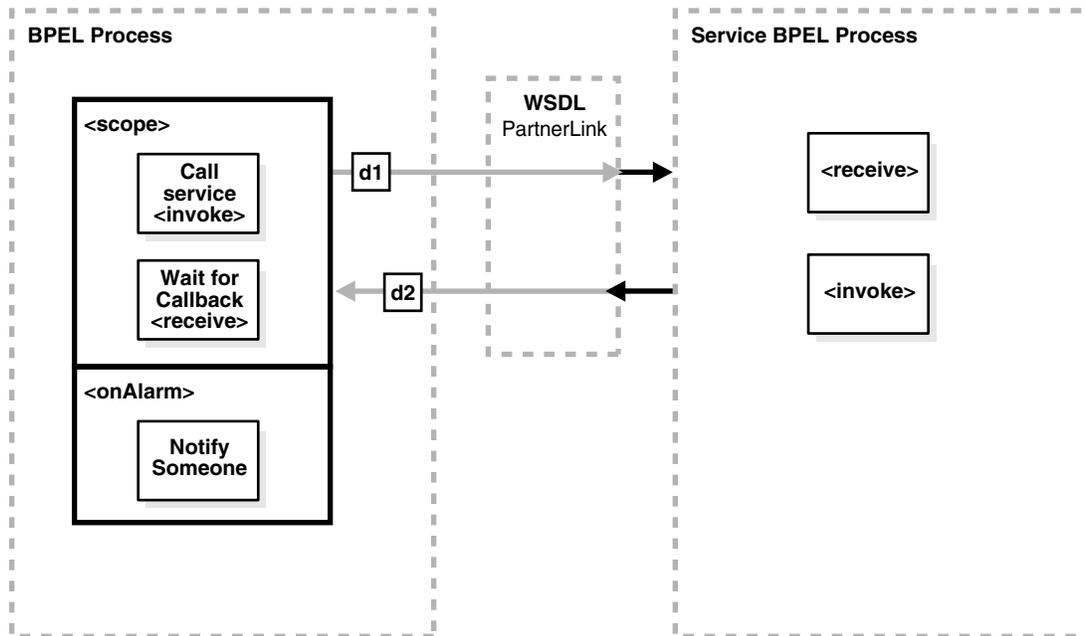
For more information about asynchronous interactions with a timeout, see [Section 14.2, "Creating a Pick Activity to Select Between Continuing a Process or Waiting."](#)

BPEL Process Service Component as the Service

The behavior of the BPEL process service component as a service is equal to the behavior with the asynchronous interaction with the BPEL process service component as the service.

6.5 Introduction to Asynchronous Interactions with a Notification Timer

In an asynchronous interaction with a notification time, a client sends a request to a service and waits for a reply, although a notification is sent after a timer expires. The client continues to wait for the reply from the service even after the timer has expired. [Figure 6–5](#) provides an overview.

Figure 6–5 Asynchronous Interaction with a Notification Time**BPEL Process Service Component as the Client**

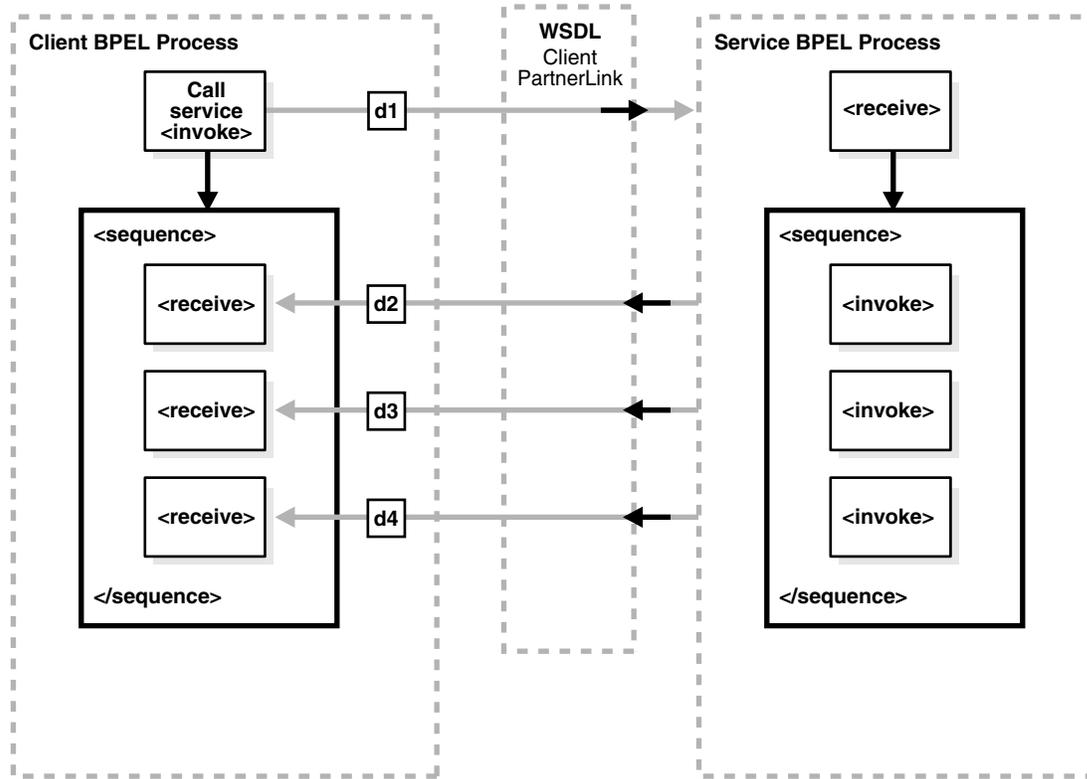
When the BPEL process service component is on the client side of this transaction, it needs a scope activity containing an invoke activity to send the request, and a receive activity to accept the reply. The onAlarm handler of the scope activity has a time limit and instructions on what to do when the timer expires. For example, wait 30 minutes, then send a warning indicating that the process is taking longer than expected. As with all partner activities, the WSDL file defines the interaction.

BPEL Process Service Component as the Service

The behavior for the BPEL process service component as the service is equal to the behavior with the asynchronous interaction with the BPEL process service component as the service.

6.6 Introduction to One Request, Multiple Responses

In this interaction type, the client sends a single request to a service and receives multiple responses in return. For example, the request can be to order a product online, and the first response can be the estimated delivery time, the second response a payment confirmation, and the third response a notification that the product has shipped. In this example, the number and types of responses are expected. [Figure 6–6](#) provides an overview.

Figure 6–6 One Request, Multiple Responses**BPEL Process Service Component as the Client**

When the BPEL process service component is on the client side of this transaction, it needs an invoke activity to send the request, and a sequence activity with three receive activities, one for each reply. As with all partner activities, the WSDL file defines the interaction.

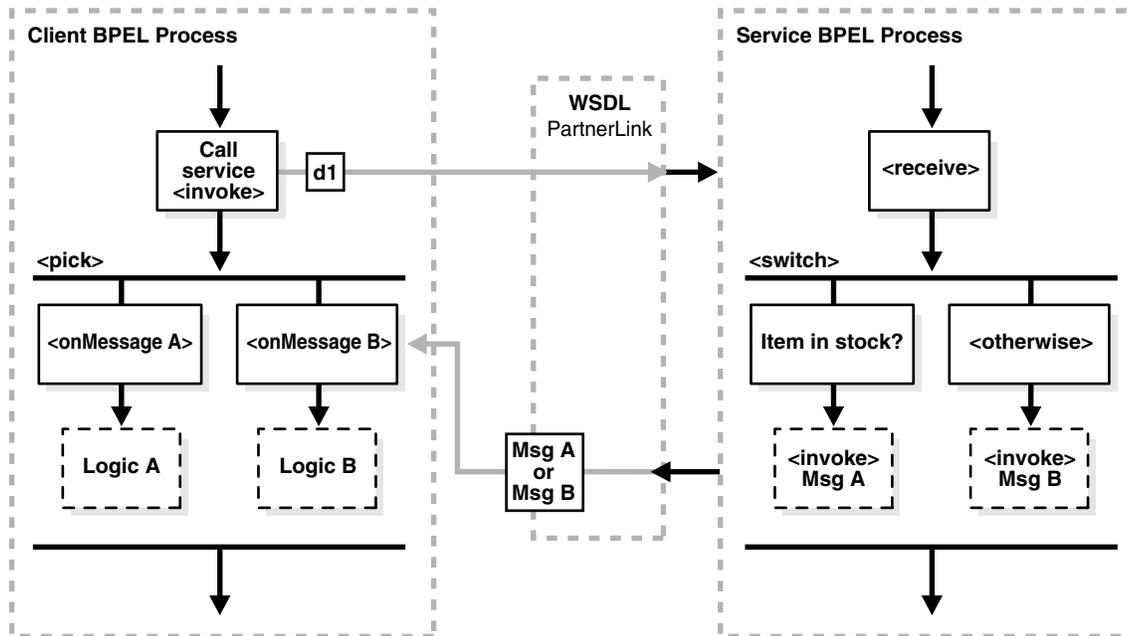
BPEL Process Service Component as the Service

The BPEL service needs a receive activity to accept the message from the client, and a sequence attribute with three invoke activities, one for each reply.

6.7 Introduction to One Request, One of Two Possible Responses

In an interaction using one request and one of two possible responses, the client sends a single request to a service and receives one of two possible responses. For example, the request can be to order a product online, and the first response can be either an in-stock message, or an out-of-stock message. [Figure 6–7](#) provides an overview.

Figure 6–7 One Request, One of Two Possible Responses



BPEL Process Service Component as the Client

When the BPEL process service component is on the client side of this transaction, it needs the following:

- An invoke activity to send the request
- A pick activity with two branches: one onMessage for the in-stock response and instructions on what to do if an in-stock message is received
- A second onMessage for the out-of-stock response and instructions on what to do if an out-of-stock message is received

As with all partner activities, the WSDL file defines the interaction.

For more information about interactions using one request and one of two possible responses, see [Section 14.2, "Creating a Pick Activity to Select Between Continuing a Process or Waiting."](#)

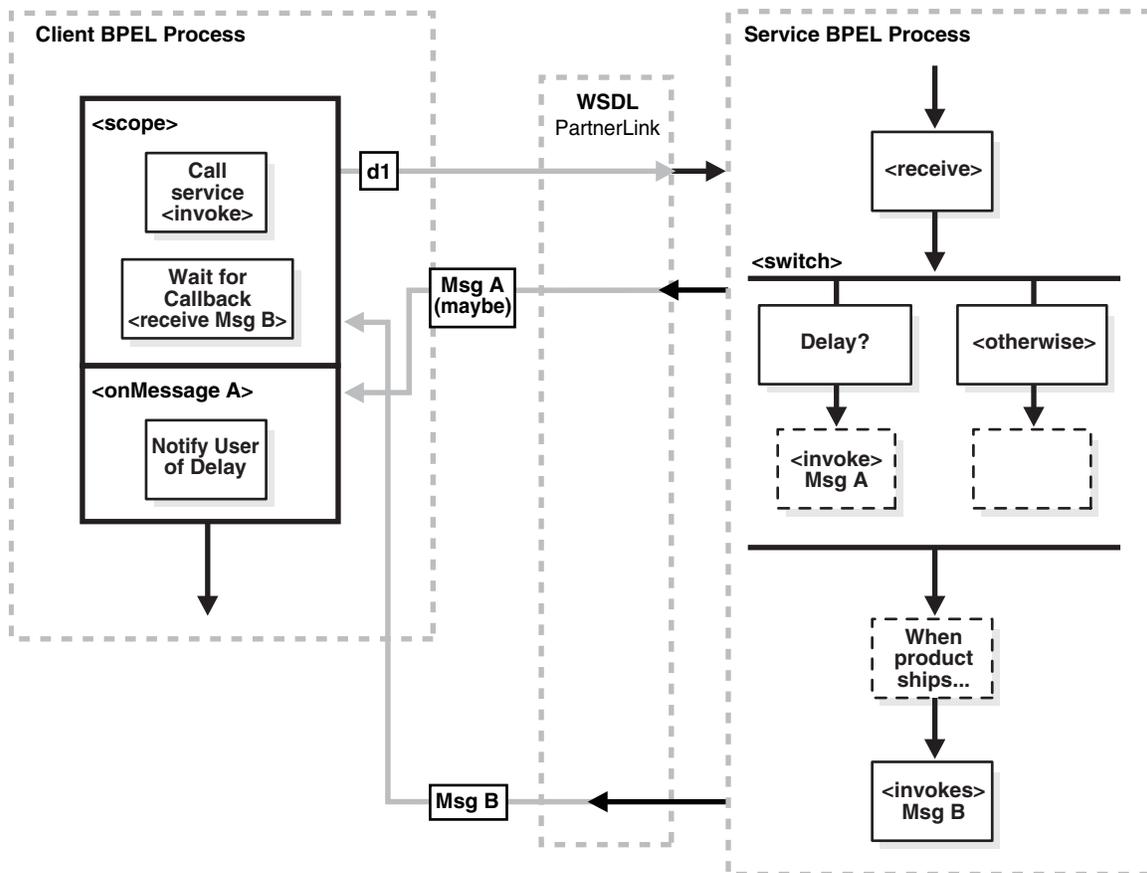
BPEL Process Service Component as the Service

The BPEL service needs a receive activity to accept the message from the client, and a switch activity with two branches, one with an invoke activity sending the in-stock message if the item is available, and a second branch with an invoke activity sending the out-of-stock message if the item is not available.

6.8 Introduction to One Request, a Mandatory Response, and an Optional Response

In this type of interaction, the client sends a single request to a service and receives one or two responses. Here, the request is to order a product online. If the product is delayed, the service sends a message letting the customer know. In any case, the service always sends a notification when the item ships. [Figure 6–8](#) provides an overview.

Figure 6–8 One Request, a Mandatory Response, and an Optional Response



BPEL Process Service Component as the Client

When the BPEL process service component is on the client side of this transaction, it needs a scope activity containing the invoke activity to send the request, and a receive activity to accept the mandatory reply. The onMessage handler of the scope activity is set to accept the optional message and instructions on what to do if the optional message is received (for example, notify you that the product has been delayed). The client BPEL process service component waits to receive the mandatory reply. If the mandatory reply is received first, the BPEL process service component continues without waiting for the optional reply. As with all partner activities, the WSDL file defines the interaction.

BPEL Process Service Component as the Service

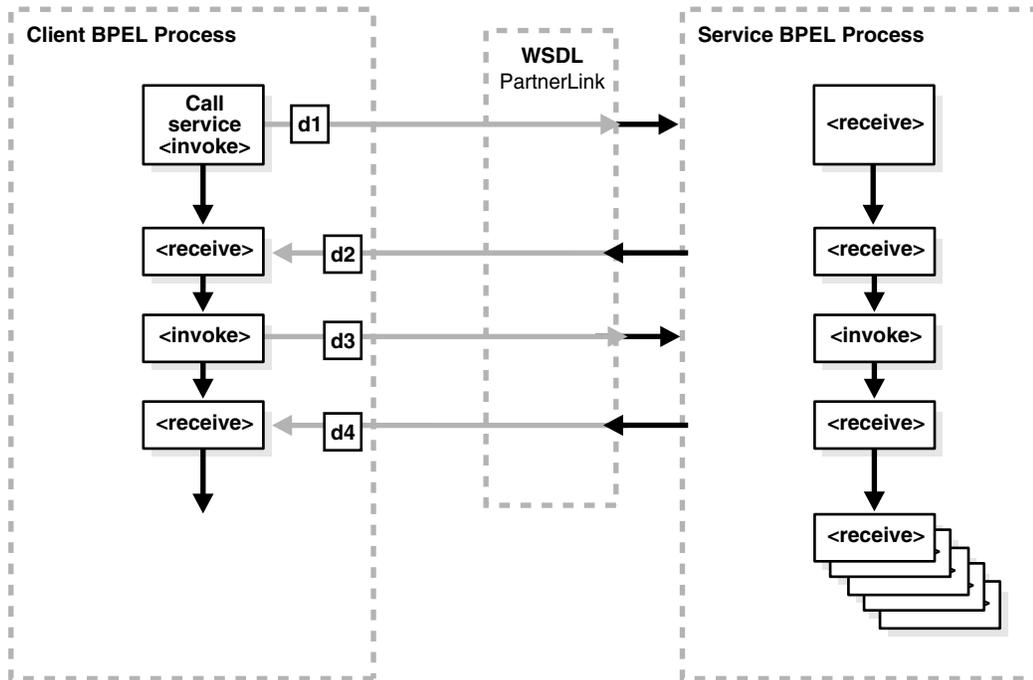
The BPEL service needs a scope activity containing the receive activity and an invoke activity to send the mandatory shipping message, and the scope's onAlarm handler to send the optional delayed message if a timer expires (for example, send the delayed message if the item is not shipped in 24 hours).

6.9 Introduction to Partial Processing

In partial processing, the client sends a request to a service and receives an immediate response, but processing continues on the service side. For example, the client sends a request to purchase a vacation package, and the service sends an immediate reply confirming the purchase, then continues on to book the hotel, the flight, the rental car,

and so on. This pattern can also include multiple short callbacks, followed by longer-term processing. [Figure 6-9](#) provides an overview.

Figure 6-9 Partial Processing



BPEL Process Service Component as the Client

In this case, the BPEL client is simple; it needs an invoke activity for each request and a receive activity for each reply for asynchronous transactions, or just an invoke activity for each synchronous transaction. Once those transactions are complete, the remaining work is handled by the service. As with all partner activities, the WSDL file defines the interaction.

BPEL Process Service Component as the Service

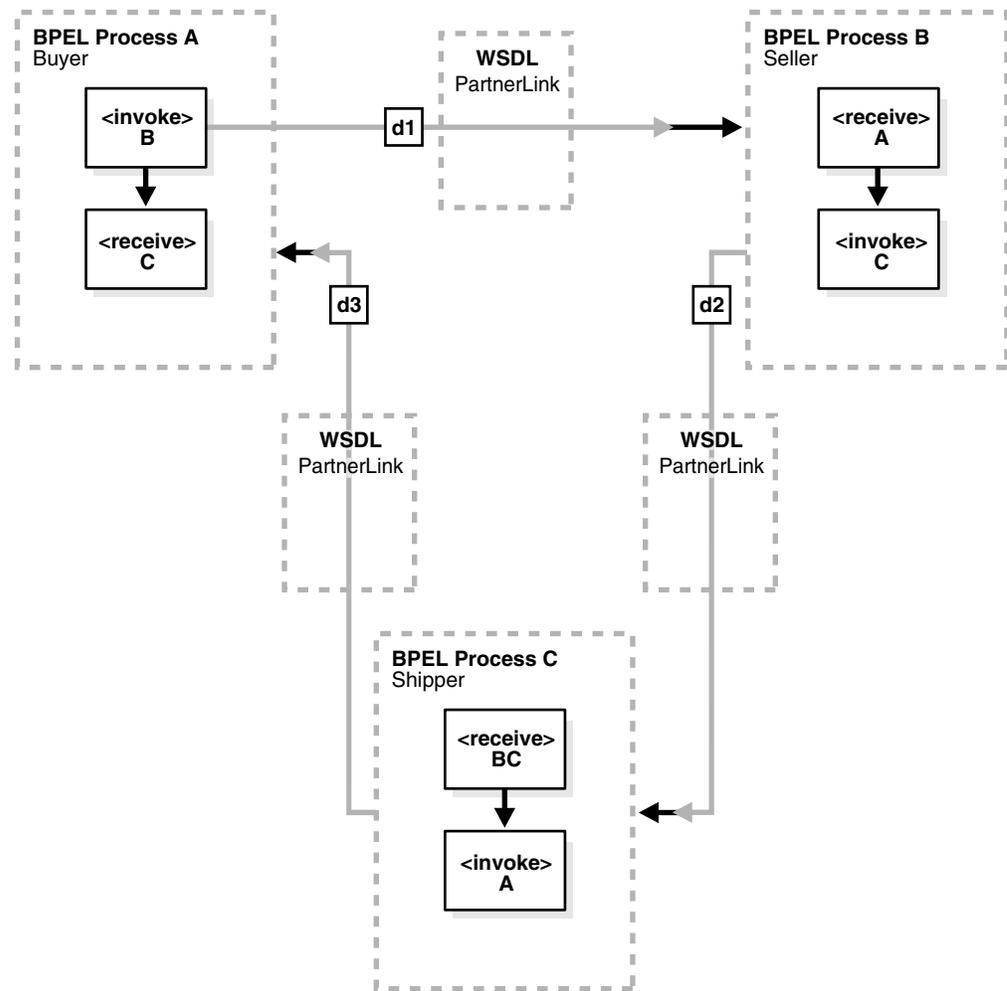
The BPEL service needs a receive activity for each request from the client, and an invoke activity for each response. Once the responses are finished, the BPEL process service component as the service can continue with its processing, using the information gathered in the interaction to perform the necessary tasks without any further input from the client.

6.10 Introduction to Multiple Application Interactions

In some cases, there are more than two applications involved in a transaction, for example, a buyer, seller, and shipper. In this case, the buyer sends a request to the seller, the seller sends a request to the shipper, and the shipper sends a notification to the buyer. This A-to-B-to-C-to-A transaction pattern can handle many transactions at the same time. Therefore, a mechanism is required for keeping track of which message goes where. [Figure 6-10](#) provides an overview.

As with all partner activities, the WSDL file defines the interaction.

Figure 6–10 Multiple Party Interactions



This kind of coordination can be managed using WS-Addressing or correlation sets. For more information about both, see [Chapter 9, "Invoking an Asynchronous Web Service from a BPEL Process."](#)

Manipulating XML Data in a BPEL Process

This chapter describes how to manipulate XML data in a BPEL process service component. This chapter provides a variety of examples. Topics include how to work with variables, sequences, and arrays, how to use XPath expressions, and how to perform tasks such as mathematical calculations. The explanations are largely by example, and provide an introduction to the supported specifications.

This chapter includes the following sections:

- [Section 7.1, "Introduction to Manipulating XML Data in BPEL Processes"](#)
- [Section 7.2, "Delegating XML Data Operations to Data Provider Services"](#)
- [Section 7.3, "Using Standalone SDO-based Variables"](#)
- [Section 7.4, "Initializing a Variable with Expression Constants or Literal XML"](#)
- [Section 7.5, "Copying Between Variables"](#)
- [Section 7.6, "Accessing Fields Within Element-Based and Message Type-Based Variables"](#)
- [Section 7.7, "Assigning Numeric Values"](#)
- [Section 7.8, "Using Mathematical Calculations with XPath Standards"](#)
- [Section 7.9, "Assigning String Literals"](#)
- [Section 7.10, "Concatenating Strings"](#)
- [Section 7.11, "Assigning Boolean Values"](#)
- [Section 7.12, "Assigning a Date or Time"](#)
- [Section 7.13, "Manipulating Attributes"](#)
- [Section 7.14, "Manipulating XML Data with bpelex Extensions"](#)
- [Section 7.15, "Validating XML Data with bpelex:validate"](#)
- [Section 7.16, "Manipulating XML Data Sequences That Resemble Arrays"](#)
- [Section 7.17, "Converting from a String to an XML Element"](#)
- [Section 7.18, "Understanding the Differences Between Document-Style and RPC-Style WSDL Files"](#)
- [Section 7.19, "Manipulating SOAP Headers in BPEL"](#)
- [Section 7.20, "Using MIME/DIME SOAP Attachments"](#)

Note: Most of the examples in this chapter assume that the WSDL file defining the associated message types is document-literal style rather than the RPC style. There is a difference in how XPath query strings are formed for RPC-style WSDL definitions. If you are working with a type defined in an RPC WSDL file, see [Section 7.18, "Understanding the Differences Between Document-Style and RPC-Style WSDL Files."](#)

7.1 Introduction to Manipulating XML Data in BPEL Processes

This section provides an introduction to using XML data in BPEL processes.

7.1.1 XML Data in BPEL

In a BPEL process service component, most pieces of data are in XML format. This includes the messages passed to and from the BPEL process service component, the messages exchanged with external services, and local variables used by the process. You define the types for these messages and variables with the XML schema, usually in the Web Services Description Language (WSDL) file for the flow, the WSDL files for the services it invokes, or the XSD file referenced by those WSDL files. Therefore, most variables in BPEL are XML data, and any BPEL process service component uses much of its code to manipulate these XML variables. This typically includes performing data transformation between representations required for different services, and local manipulation of data (for example, to combine the results from several service invocations).

BPEL also supports service data object (SDO) variables, which are not in an XML format, but rather in a memory structure format.

7.1.2 Data Manipulation and XPath Standards

The starting point for data manipulation in BPEL is the assign activity, which builds on the XPath standard. XPath queries, expressions, and functions play a large part in this type of manipulation. In addition, more advanced methods are available that involve using XQuery, XSLT, or Java, usually to do more complex data transformation or manipulation.

This section provides a general overview of how to manipulate XML data in BPEL. It summarizes the key building blocks used in various combinations and provides examples. The remaining sections in this chapter discuss and illustrate how to apply these building blocks to perform specific tasks.

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. 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. [Example 7-1](#) shows the formal syntax, as described in the *Business Process Execution Language for Web Services Specification*:

Example 7-1 Assign Activity

```
<assign standard-attributes>
  standard-elements
  <copy>
    from-spec
    to-spec
  </copy>
```

```
</assign>
```

This syntax is described in detail in that specification. The `from-spec` and `to-spec` typically specify a variable or variable part, as shown in [Example 7-2](#):

Example 7-2 from-spec and to-spec Attributes

```
<assign>
  <copy>
    <from variable="c1" part="address"/>
    <to variable="c3"/>
  </copy>
</assign>
```

When you use Oracle JDeveloper, you supply assign activity details in a Copy Operation dialog that includes a **From** section and a **To** section. This reflects the preceding BPEL source code syntax.

XPath standards play a key role in the assign activity. Brief examples are shown here as an introduction; examples with more context and explanation are provided in the sections that follow.

- 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.

[Example 7-3](#) provides an example:

Example 7-3 query Attribute

```
<from variable="input" part="payload"
  query="/p:CreditFlowRequest/p:ssn"/>
```

The value of the query attribute must be a location path that selects exactly one node. You can find further details about the query attribute and XPath standards syntax in the *Business Process Execution Language for Web Services Specification* (section 14.3) and the *XML Path Language (XPath) Specification*, respectively.

- XPath expressions

You use an XPath expression (specified in an `expression` attribute in 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). Similarly, the value of an expression attribute must return exactly one node or one object only when it is used in the `from` clause within a copy operation. For more information about XPath expressions, see section 9.1.4 of the *XML Path Language (XPath) Specification*.

Within XPath expressions, you can call the following types of functions:

- Core XPath functions

XPath supports a large number of built-in functions, including functions for string manipulation (such as `concat`), numeric functions (like `sum`), and others.

```
<from expression="concat('string one', 'string two')"/>
```

For a complete list of the functions built into XPath standards, see section 4 of the *XML Path Language (XPath) Specification*.

- BPEL XPath extension functions

BPEL adds several extension functions to the core XPath core functions, enabling XPath expressions to access information from a process. The extensions are defined in the standard BPEL namespace

<http://schemas.xmlsoap.org/ws/2003/03/business-process/> and indicated by the prefix `bpws`:

```
<from expression= "bpws:getVariableData('input', 'payload', '/p:value') + 1"/>
```

For more information, see sections 9.1 and 14.1 of the *Business Process Execution Language for Web Services Specification*.

- Oracle BPEL XPath extension functions

Oracle provides some additional XPath functions that use the capabilities built into BPEL and XPath standards for adding new functions.

These functions are defined in the namespace

<http://schemas.oracle.com/xpath/extension> and indicated by the prefix `ora:`.

- Custom functions

Oracle BPEL Process Manager functions are defined in the

`bpel-xpath-functions-config.xml` and placed inside the `orabpel.jar` file. For more information, see [Section B.7, "Creating User-Defined XPath Extension Functions"](#) and *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite*.

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, Java, or a `bpelx` operation under an assign activity (See [Section 7.14, "Manipulating XML Data with `bpelx` Extensions"](#)), or as a web service. For XSLT, Oracle BPEL Process Manager includes XPath functions that execute these transformations.

For more information about XPath and XQuery transformation code examples, see [Chapter 45, "Creating Transformations with the XSLT Mapper."](#)

Note: Passing large schemas through an assign activity can cause Oracle JDeveloper to freeze up and run low on memory if you right-click the payload in the **From** or **To** section of the Copy Operation dialog and select **Expand All**. As a workaround, manually expand the payload elements.

7.2 Delegating XML Data Operations to Data Provider Services

You can specify BPEL data operations to be performed by an underlying data provider service through use of the entity variable. The data provider service performs the data operations in a data store behind the scenes and without use of other data store-related features provided by Oracle SOA Suite (for example, the database adapter). This action enhances Oracle SOA Suite runtime performance and incorporates native features of the underlying data provider service during compilation and runtime.

For this release, the entity variable can be used with an Oracle Application Development Framework (ADF) Business Component data provider service using SDO-based data.

In previous releases, variables and messages exchanged within a BPEL business process were disconnected payload (a snapshot of data returned by a web service) placed into an XML structure. In some cases, the user required this type of fit. In other cases, this fit presented challenges.

The entity variable addresses the following challenges of previous releases:

- Extensive data conversion
If the underlying data was not in XML form, data conversion (for example, translating delimited text to XML) was required. If the underlying size of the data was large, the processing potentially impacted performance.
- Stale snapshot data
Variables (including WSDL messages) in BPEL processes were disconnected payload. In some cases, this was required. In other cases, you wanted a variable to represent the most recent data being modified by other applications outside Oracle BPEL Process Manager. This meant the disconnected data model provided a stale data set that did not fit all needs. The snapshot also duplicated data, which impacted performance when the data size was large.
- Loss of native data behavior
Some data conversion implementation required data structure enforcement or business data logic beyond the XML schema. For example, the start date needed to be smaller than the end date. When the variable was a disconnected payload, validation occurred only during related web service invocation. The need to optionally perform the extra business data logic after certain operations, but before web service invocation, was sometimes preferred.

To address these challenges with this release, you create an entity variable during variable declaration. An entity variable acts as a data handle to access and plug in different data provider service technologies behind the scenes. During compilation and runtime, Oracle BPEL Process Manager delegates data operations to the underlying data provider service.

[Table 7-1](#) provides an example of how data conversion was performed in previous releases (using the database adapter as an example) and in release 11g with the entity variable.

Table 7-1 Data Manipulation Capabilities in Previous and Current Releases

10.1.x Releases	11g Release When Using the Entity Variable
Data operations such as explicitly loading and saving data were performed by the database adapter in Oracle BPEL Process Manager. All data (for example, of a purchase order) was saved in the database dehydration store.	Data operations such as loading and saving data are performed automatically by the data provider service (the Oracle ADF Business Component application), without asking you to code any service invocation.
	Oracle BPEL Process Manager stores a key (for example, purchase order ID (POID)) that points to this data. Oracle BPEL Process Manager fetches the key when access to data is requested (the bind entity activity does this). You must explicitly request the data to be bound using the key. Any data changes are persisted by the data provider service in a database that can be different from the dehydration store database. This prevents data duplication.

Table 7–1 (Cont.) Data Manipulation Capabilities in Previous and Current Releases

10.1.x Releases	11g Release When Using the Entity Variable
Data in variables was in document object model (DOM) form	Data in variables is in SDO form, which provides for a simpler conversion process than DOM, especially when the data provider service understands SDO forms.

Note: Only BPEL process service components currently allow the use of SDO-formed variables. If your composite application has an Oracle Mediator service component wired with an SDO-based Java binding component reference, the data form of the variable defaults to DOM. In addition, the features described for 10.1.x releases in [Table 7–1](#) are still supported in release 11g.

The following documentation describes use of the entity variable:

- [bpel-203-EntityVariableToADFBC](#):
This sample uses an entity variable bound to an Oracle ADF BC service using an SDO interface. This provides the BPEL process with a variable that behaves like a standard BPEL variable. However, the data is maintained outside the BPEL process (in this case, in an Oracle ADF BC component). Rather than passing around a large payload of data, it resides in one place. A reference key is passed around to read and update the data.
- [bpel-204-EntityVariableToBPELBackedSDO](#):
This sample shows how you can use the Oracle ADF BC SDO interface, but with a back-end implementation other than an Oracle ADF BC application. In this case, the back end is implemented using a BPEL process.
Visit the following URL for details about these samples:
http://www.oracle.com/technology/sample_code/products/bpel
- *Oracle Fusion Middleware Tutorial for Running and Building an Application with Oracle SOA Suite* describes how to create an entity variable

7.2.1 How to Create an Entity Variable

This section describes how to create an entity variable and a binding key in Oracle JDeveloper.

In previous releases of Oracle BPEL Process Manager, all variable data was in DOM form. With release 11g, variable data in SDO form is also supported. DOM and SDO variables in BPEL process service components are implicitly converted to the required forms. For example, an Oracle BPEL process service component using DOM-based variables can automatically convert these variables as required to SDO-based variables in an assign activity, and vice versa. Both form types are defined in the XSD schema file. No user intervention is required.

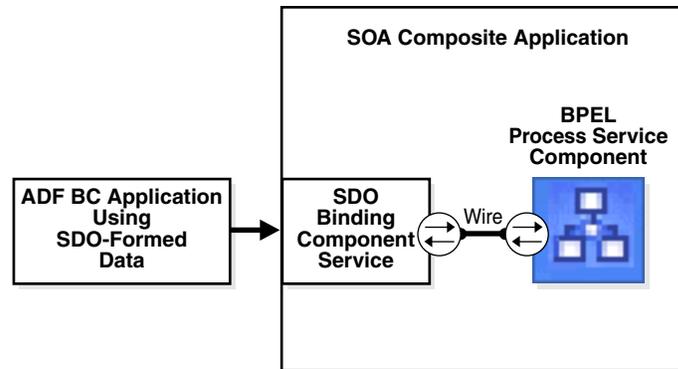
Entity variables also support SDO-formed data. However, unlike the DOM and SDO variables, the entity variable with SDO-based data enables you to bind a unique key value to data (for example, a purchase order). Only the key is stored in the dehydration store; the data requiring conversion is stored with the service of the Oracle ADF Business Component application. The key points to the data stored in the service. When the data is required, it is fetched from the data provider service and

placed into memory. The process occurs in two places: the bind entity activity and the dehydration store. For example, when Oracle BPEL Process Manager rehydrates, it stores only the key for the entity variable; when it wakes up, it does an implicit bind to get the current data.

7.2.1.1 Understanding How SDO Works in the Inbound Direction

The SDO binding component service provides the outside world with an entry point to the composite application, as shown in [Figure 7-1](#).

Figure 7-1 Inbound Direction



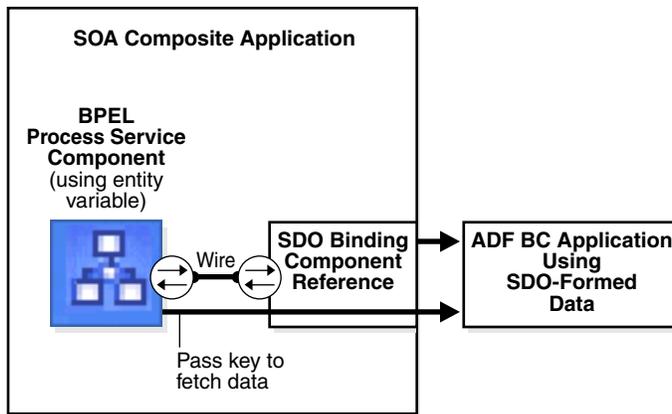
You use the SOA Composite Editor and Oracle BPEL Designer to perform the following tasks:

- Define an SDO binding component service and a BPEL process service component in the composite application.
- Connect (wire) the SDO service and BPEL process service component.
- Define the details of the BPEL process service component.

For more information about using the SOA Composite Editor, see [Chapter 4, "Introduction to the Functionality of the SOA Composite Editor."](#)

7.2.1.2 Understanding How SDO Works in the Outbound Direction

The SDO binding component reference enables messages to be sent from the composite application to Oracle ADF Business Component application external partners in the outside world, as shown in [Figure 7-2](#).

Figure 7-2 Outbound Direction

When the Oracle ADF Business Component application is the external partner link to the outside world, there is no SDO binding component reference in the SOA Composite Editor that you drag into the composite application to create outbound communication. Instead, communication between the composite application and the Oracle ADF Business Component application occurs as follows:

- The Oracle ADF Business Component application is deployed and automatically registered as an SDO service in the Service Infrastructure
- Oracle JDeveloper is used to browse for and discover this application as an ADF-BC service and create a partner link connection.
- The `composite.xml` file is automatically updated with reference details (the `binding.adf` property) when the Oracle ADF Business Component application service is discovered.

7.2.1.3 Creating an Entity Variable and Choosing a Partner Link

You now create an entity variable and select a partner link for the Oracle ADF Business Component application. The following example describes how the OrderProcessor BPEL process service component receives an ID for an order by using a bind entity activity to point to order data in an Oracle ADF Business Component data provider service in the WebLogic Fusion Order Demo application.

For more information, see *Oracle Fusion Middleware Tutorial for Running and Building an Application with Oracle SOA Suite*.

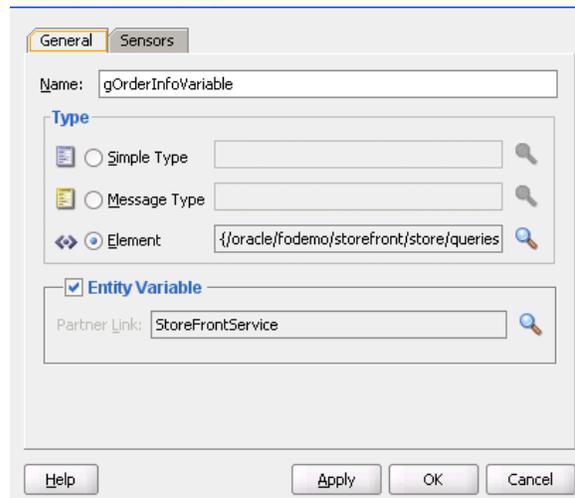
To create an entity variable and choose a partner link:

1. Go to the Structure window of the BPEL process service component in Oracle JDeveloper.
2. Right-click the **Variables** folder and select **Expand All Child Nodes**.
3. In the second **Variables** folder, right-click and select **Create Variable**.
The Create Variable dialog appears.
4. In the **Name** field, enter a name.
5. Click the **Entity Variable** checkbox and select the **Search** icon to the right of the **Partner Link** field.

The Partner Link Chooser dialog appears with a list of available services, including the SDO service called **ADF-BC Service**.

6. Browse for and select the service for the Oracle ADF Business Component application.
7. Click **OK** to close the Partner Link Chooser and Create Variable dialogs.
The Create Variable dialog looks as shown in [Figure 7-3](#).

Figure 7-3 Create Variable Dialog



7.2.1.4 Creating a Binding Key

You now create a key to point to the order data in the Oracle ADF Business Component data provider service.

To create a binding key:

1. Drag a **Bind Entity** activity into your BPEL process service component.
The Bind Entity dialog appears.
2. In the **Name** field, enter a name.
3. To the right of the **Entity Variable** field, click the **Search** icon.
The Variable Chooser dialog appears.
4. Select the entity variable created in [Section 7.2.1.3, "Creating an Entity Variable and Choosing a Partner Link"](#) and click **OK**.
5. In the **Unique Keys** section, click the **Add** icon.
The Specify Key dialog appears. You use this dialog to create a key for retrieving the order ID from the Oracle ADF Business Component data provider service.
6. Enter the details described in [Table 7-2](#) to define the binding key:

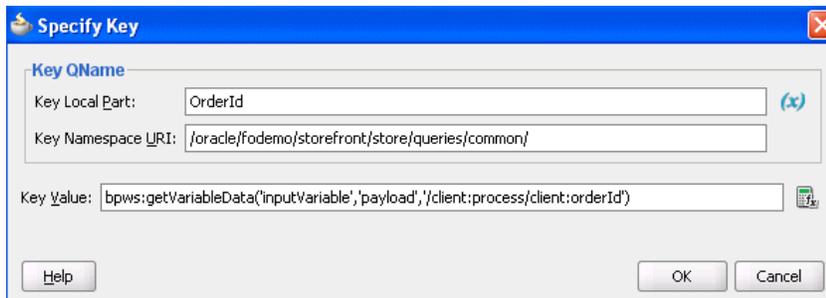
Table 7-2 Specify Key Dialog Fields and Values

Field	Value
Key Local Part	Enter the local part of the key.
Key Namespace URI	Enter the namespace URI for the key.

Table 7–2 (Cont.) Specify Key Dialog Fields and Values

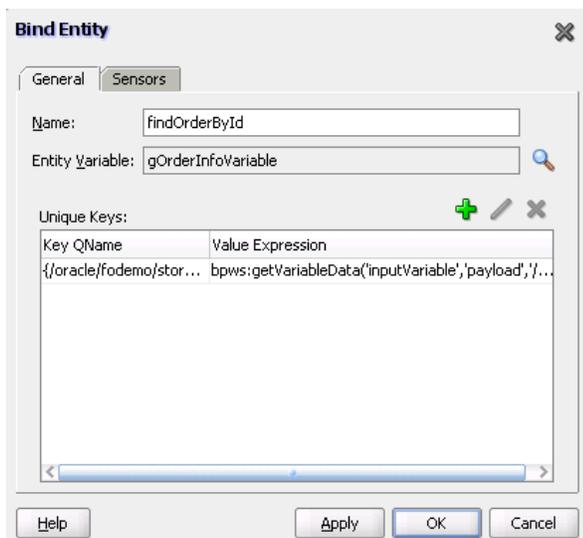
Field	Value
Key Value	<p>Enter the key value expression. This expression must match the type of a key. The following examples show expression value keys for a POID key:</p> <ul style="list-style-type: none"> ▪ <code>\$inputMsg.payload/tns:poid</code> ▪ <code>bpws:getVariableData('inputmsg','payload','tns:poid')</code> <p>The POID key for an entity variable typically comes from another message. If the type of POID key is an integer and the expression result is a string of ABC, the string-to-integer fails and the bind entity activity also fails at runtime.</p>

Figure 7–4 shows the Specify Key dialog after completion.

Figure 7–4 Specify Key Dialog

7. Click **OK** to close the Specify Key dialog.

A name-pair value appears in the **Unique Keys** table, as shown in Figure 7–5. Design is now complete.

Figure 7–5 Bind Entity Dialog

8. Click **OK** to close the Bind Entity dialog.

After the Bind Entity activity is executed at runtime, the entity variable is ready to be used.

For more information about using SDOs, see *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*. This guide describes how to expose application modules as web services and publish rows of view data objects as SDOs. The application module is the ADF framework component that encapsulates business logic as a set of related business functions.

7.3 Using Standalone SDO-based Variables

Standalone SDO-based variables are similar to ordinary BPEL XML-DOM-based variables. The major difference is that the underlying data form is SDO-based, instead of DOM-based. Therefore, SDO-based variables can use some SDO features such as Java API access, an easier-to-use update API, and the change summary. However, SDO usage is also subject to some restrictions that do not exist with XML-DOM-based variables. The most noticeable restriction is that SDO only supports a small subset of XPath expressions.

7.3.1 How to Declare SDO-based Variables

The syntax for declaring an SDO-based variable is similar to that for declaring BPEL variables. [Example 7-4](#) provides details.

Example 7-4 SDO-based Variable Declaration

```
<variable name="deptVar_s" element="hrtypes:dept" />
<variable name="deptVar_v" element="hrtypes:dept" bpelx:sdoCapable="false" />
```

If you want to override the automatic detection, use the `bpelx:sdoCapable="true|false"` switch. For example, variable `deptVar_v` described in [Example 7-4](#) is a regular DOM-based variable. [Example 7-4](#) provides an example of the schema.

Example 7-5 XSD Sample

```
<xsd:element name="dept" type="Dept" />
<xsd:complexType name="Dept"
  sdoJava:instanceClass="sdo.sample.service.types.Dept">
  <xsd:annotation>
    <xsd:appinfo source="Key"
      xmlns="http://xmlns.oracle.com/bc4j/service/metadata/">
      <key>
        <attribute>Deptno</attribute>
      </key>
      <fetchMode>minimal</fetchMode>
    </xsd:appinfo>
  </xsd:annotation>
  <xsd:sequence>
    <xsd:element name="Deptno" type="xsd:integer" minOccurs="0" />
    <xsd:element name="Dname" type="xsd:string" minOccurs="0"
      nillable="true" />
    <xsd:element name="Loc" type="xsd:string" minOccurs="0" nillable="true" />
    <xsd:element name="Emp" type="Emp" minOccurs="0" maxOccurs="unbounded"
      nillable="true" />
  </xsd:sequence>
</xsd:complexType>
```

7.3.2 How to Convert from XML to SDO

Oracle BPEL Process Manager supports dual data forms: DOM and SDO. You can interchange the usage of DOM-based and SDO-based variables within the same business process, even within the same expression. The Oracle BPEL Process Manager data framework automatically converts back and forth between DOM and SDO forms.

By using the entity variable XPath rewrite capabilities, Oracle BPEL Process Manager enables some XPath features (for example, variable reference and function calls) that the basic SDO specification does not support. However, there are other limitations on the XPath used with SDO-based variables (for example, there is no support for `and`, `or`, and `not`).

[Example 7–6](#) provides a simple example of converting from XML to SDO.

Example 7–6 XML-to-SDO Conversion

```
<assign>
  <copy>
    <from>
      <ns0:dept xmlns:ns0="http://sdo.sample.service/types/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
        <ns0:Deptno>10</ns0:Deptno>
        <ns0:Dname>ACCOUNTING</ns0:Dname>
        <ns0:Loc>NEW YORK</ns0:Loc>
        <ns0:Emp>
          <ns0:Empno>7782</ns0:Empno>
          <ns0:Ename>CLARK</ns0:Ename>
          <ns0:Job>MANAGER</ns0:Job>
          <ns0:Mgr>7839</ns0:Mgr>
          <ns0:Hiredate>1981-06-09</ns0:Hiredate>
          <ns0:Sal>2450</ns0:Sal>
          <ns0:Deptno>10</ns0:Deptno>
        </ns0:Emp>
        <ns0:Emp>
          <ns0:Empno>7839</ns0:Empno>
          <ns0:Ename>KING</ns0:Ename>
          <ns0:Job>PRESIDENT</ns0:Job>
          <ns0:Hiredate>1981-11-17</ns0:Hiredate>
          <ns0:Sal>5000</ns0:Sal>
          <ns0:Deptno>10</ns0:Deptno>
        </ns0:Emp>
        <ns0:Emp>
          <ns0:Empno>7934</ns0:Empno>
          <ns0:Ename>MILLER</ns0:Ename>
          <ns0:Job>CLERK</ns0:Job>
          <ns0:Mgr>7782</ns0:Mgr>
          <ns0:Hiredate>1982-01-23</ns0:Hiredate>
          <ns0:Sal>1300</ns0:Sal>
          <ns0:Deptno>10</ns0:Deptno>
        </ns0:Emp>
      </ns0:dept>
    </from>
    <to variable="deptVar_s" />
  </copy>
</assign>
```

[Example 7–7](#) provides an example of copying from an XPath expression of an SDO variable to a DOM variable.

Example 7-7 Copy from an XPath Expression of an SDO Variable to a DOM Variable

```

<assign>
  <!-- copy from an XPath expression of an SDO variable to DOM variable -->
  <copy>
    <from expression="$deptVar_s/hrtypes:Emp[2]" />
    <to variable="empVar_v" />
  </copy>
  <!-- copy from an XPath expression of an DOM variable to SDO variable -->
  <copy>
    <from expression="$deptVar_v/hrtypes:Emp[2]" />
    <to variable="empVar_s" />
  </copy>
  <!-- insert a DOM based data into an SDO variable -->
  <bpelx:insertAfter>
    <bpelx:from variable="empVar_v" />
    <bpelx:to variable="deptVar_s" query="hrtypes:Emp" />
  </bpelx:insertAfter>
  <!-- insert a SDO based data into an SDO variable at particular location,
       no XML conversion is needed -->
  <bpelx:insertBefore>
    <bpelx:from expression="$deptVar_s/hrtypes:Emp[hrtypes:Sal = 1300]" />
    <bpelx:to variable="deptVar_s" query="hrtypes:Emp[6]" />
  </bpelx:insertBefore>
</assign>

```

Example 7-8 provides an example of removing a portion of SDO data.

Example 7-8

```

<assign>
  <bpelx:remove>
    <bpelx:target variable="deptVar_s" query="hrtypes:Emp[2]" />
  </bpelx:remove>
</assign>

```

Note: The `bpelx:append` operation is not supported for SDO-based variables for the following reasons:

- The `<copy>` operation on an SDO-based variable has smart update capabilities (for example, you do not need to perform a `<bpelx:append>` before the `<copy>` operation).
 - The SDO data object is metadata driven and does not generally support adding a new property arbitrarily.
-
-

7.4 Initializing a Variable with Expression Constants or Literal XML

It is often useful to assign literal XML to a variable in BPEL, for example, to initialize a variable before copying dynamic data into a specific field within the XML data content for the variable. This is also useful for testing purposes when you want to hard code XML data values into the process.

7.4.1 How To Assign a Literal XML Element

Example 7-9 assigns a literal `result` element to the payload part of the output variable:

Example 7–9 Literal Element Assignment

```

<assign>
  <!-- copy from literal xml to the variable -->
  <copy>
    <from>
      <result xmlns="http://samples.otn.com">
        <name/>
        <symbol/>
        <price>12.3</price>
        <quantity>0</quantity>
        <approved/>
        <message/>
      </result>
    </from>
    <to variable="output" part="payload" />
  </copy>
</assign>

```

7.5 Copying Between Variables

When you copy between variables, you copy directly from one variable (or part) to another variable of a compatible type, without needing to specify a particular field within either variable. In other words, there is no need to specify an XPath query.

7.5.1 How to Copy Between Variables

[Example 7–10](#) shows two assignments being performed, first copying between two variables of the same type and then copying a variable part to another variable with the same type as that part.

Example 7–10 Copying Between Variables

```

<assign>
  <copy>
    <from variable="c1" />
    <to variable="c2" />
  </copy>
  <copy>
    <from variable="c1" part = "address" />
    <to variable="c3" />
  </copy>
</assign>

```

The BPEL file defines the variables shown in [Example 7–11](#):

Example 7–11 Variable Definition

```

<variable name="c1" messageType="x:person" />
<variable name="c2" messageType="x:person" />
<variable name="c3" element="y:address" />

```

The WSDL file defines the person message type shown in [Example 7–12](#):

Example 7–12 Message Type Definition

```

<message name="person" xmlns:x="http://tempuri.org/bpws/example">
  <part name="full-name" type="xsd:string" />
  <part name="address" element="x:address" />
</message>

```

For more information about this code example, see Section 9.3.2 of the *Business Process Execution Language for Web Services Specification*.

7.6 Accessing Fields Within Element-Based and Message Type-Based Variables

Given the types of definitions present in most WSDL and XSD files, you must go down to the level of copying from or to a field within part of a variable based on the element and message type. This in turn uses XML schema complex types. To perform this action, you specify an XPath query in the `from` or `to` clause of the assign activity.

7.6.1 How to Access Fields Within Element-Based and Message Type-Based Variables

In [Example 7–13](#), the `ssn` field is copied from the `CreditFlow` process's input message into the `ssn` field of the credit rating service's input message.

Example 7–13 Field Copying Levels

```
<assign>
  <copy>
    <from variable="input" part="payload"
      query="/tns:CreditFlowRequest/tns:ssn" />
    <to variable="crInput" part="payload" query="/tns:ssn" />
  </copy>
</assign>
```

[Example 7–14](#) shows how the BPEL file defines message type-based variables involved in this assignment:

Example 7–14 BPEL File Definition - Message Type-Based Variables

```
<variable name="input" messageType="tns:CreditFlowRequestMessage" />
<variable name="crInput"
  messageType="services:CreditRatingServiceRequestMessage" />
```

The `crInput` variable is used as an input message to a credit rating service. Its message type, `CreditFlowRequestMessage`, is defined in the `CreditFlowService.wsdl` file, as shown in [Example 7–15](#):

Example 7–15 CreditFlowRequestMessage Definition

```
<message name="CreditFlowRequestMessage">
  <part name="payload" element="tns:CreditFlowRequest" />
</message>
```

`CreditFlowRequest` is defined with a field named `ssn`. The message type `CreditRatingServiceRequestMessage` is defined in the `CreditRatingService.wsdl` file, as shown in [Example 7–16](#):

Example 7–16 CreditRatingServiceRequestMessage Definition

```
<message name="CreditRatingServiceRequestMessage">
  <part name="payload" element="tns:ssn" />
</message>
```

A BPEL process can also use element-based variables. In [Example 7-17](#), the `autoloan` field is copied from the loan application process's input message into the `customer` field of a web service's input message.

Example 7-17 Field Copying Levels

```
<assign>
  <copy>
    <from variable="input" part="payload"
      query="/tns:invalidLoanApplication/autoloan:
        application/autoloan:customer"/>
    <to variable="customer"/>
  </copy>
</assign>
```

[Example 7-18](#) shows how the BPEL file defines element-based variables involved in an assignment:

Example 7-18 BPEL File Definition - Element-Based Variables

```
<variable name="customer" element="tns:customerProfile"/>
```

7.7 Assigning Numeric Values

You can assign numeric values in XPath expressions.

7.7.1 How to Assign Numeric Values

[Example 7-19](#) shows how to assign an XPath expression with the integer value of 100.

Example 7-19 XPath Expression Assignment

```
<assign>
  <!-- copy from integer expression to the variable -->
  <copy>
    <from expression="100"/>
    <to variable="output" part="payload" query="/p:result/p:quantity"/>
  </copy>
</assign>
```

7.8 Using Mathematical Calculations with XPath Standards

You can use simple mathematical expressions like the one in [Section 7.8.1, "How To Use Mathematical Calculations with XPath Standards,"](#) which increment a numeric value.

7.8.1 How To Use Mathematical Calculations with XPath Standards

In [Example 7-20](#), the BPEL XPath function `getVariableData` retrieves the value being incremented. The arguments to `getVariableData` are equivalent to the variable, part, and query attributes of the `from` clause (including the last two arguments, which are optional).

Example 7-20 XPath Function `getVariableData` Retrieval of a Value

```
<assign>
  <copy>
    <from expression="bpws:getVariableData('input', 'payload',
```

```

        '/p:value') + 1"/>
    <to variable="output" part="payload" query="/p:result"/>
  </copy>
</assign>

```

You can also use `$variable` syntax, as shown in [Example 7-21](#):

Example 7-21 *\$variable* Syntax Use

```

<assign>
  <copy>
    <from expression="$input.payload + 1"/>
    <to variable="output" part="payload" query="/p:result"/>
  </copy>
</assign>

```

7.9 Assigning String Literals

You can assign string literals to a variable in BPEL.

7.9.1 How to Assign String Literals

The code in [Example 7-22](#) copies an expression evaluating from the string literal 'GE' to the symbol field within the indicated variable part. (Note the use of the double and single quotes.)

Example 7-22 *Expression Copy*

```

<assign>
  <!-- copy from string expression to the variable -->
  <copy>
    <from expression="'GE'"/>
    <to variable="output" part="payload" query="/p:result/p:symbol"/>
  </copy>
</assign>

```

7.10 Concatenating Strings

Rather than copying the value of one string variable (or variable part or field) to another, you can first perform string manipulation, such as concatenating several strings.

7.10.1 How to Concatenate Strings

The concatenation is accomplished with the core XPath function named `concat`; in addition, the variable value involved in the concatenation is retrieved with the BPEL XPath function `getVariableData`. In [Example 7-23](#), `getVariableData` fetches the value of the name field from the input variable's payload part. The string literal 'Hello ' is then concatenated to the beginning of this value.

Example 7-23 *XPath Function getVariableData Fetch of Data*

```

<assign>
  <!-- copy from XPath expression to the variable -->
  <copy>
    <from expression="concat('Hello ',
      bpws:getVariableData('input', 'payload', '/p:name'))"/>
    <to variable="output" part="payload" query="/p:result/p:message"/>
  </copy>
</assign>

```

```
</copy>  
</assign>
```

Other string manipulation functions available in XPath are listed in section 4.2 of the *XML Path Language (XPath) Specification*.

7.11 Assigning Boolean Values

You can assign boolean values with the XPath boolean function.

7.11.1 How to Assign Boolean Values

[Example 7–24](#) provides an example of assigning boolean values. The XPath expression in the `from` clause is a call to XPath's boolean function `true`, and the specified approved field is set to `true`. The function `false` is also available.

Example 7–24 Boolean Value Assignment

```
<assign>  
  <!-- copy from boolean expression function to the variable -->  
  <copy>  
    <from expression="true()"/>  
    <to variable="output" part="payload" query="/result/approved"/>  
  </copy>  
</assign>
```

The XPath specification recommends that you use the `true()` and `false()` functions as a method for returning boolean constant values.

If you instead use `boolean(true)` or `boolean(false)`, the `true` or `false` inside the boolean function is interpreted as a relative element step, and not as any `true` or `false` constant. It attempts to select a child node named `true` under the current XPath context node. In most cases, the `true` node does not exist. Therefore, an empty result node set is returned and the `boolean()` function in XPath 1.0 converts an empty node set into a false result. This result can be potentially confusing.

7.12 Assigning a Date or Time

You can assign the current value of a date or time field by using the Oracle BPEL XPath function `getCurrentDate`, `getCurrentTime`, or `getCurrentDateTime`, respectively. In addition, if you have a date-time value in the standard XSD format, you can convert it to characters more suitable for output by calling the Oracle BPEL XPath function `formatDate`.

For related information, see section 9.1.2 of the *Business Process Execution Language for Web Services Specification*.

7.12.1 How to Assign a Date or Time

[Example 7–25](#) shows an example that uses the function `getCurrentDate`.

Example 7–25 Date or Time Assignment

```
<!-- execute the XPath extension function getCurrentDate() -->  
<assign>  
  <copy>  
    <from expression="xpath20:getCurrentDate()"/>  
    <to variable="output" part="payload" />
```

```

        query="/invoice/invoiceDate"/>
    </copy>
</assign>

```

In [Example 7–26](#), the `formatDate` function converts the date-time value provided in XSD format to the string 'Jun 10, 2005' (and assigns it to the string field `formattedDate`).

Example 7–26 formatDate Function

```

<!-- execute the XPath extension function formatDate() -->
<assign>
  <copy>
    <from expression="ora:formatDate('2005-06-10T15:56:00',
      'MMM dd, yyyy')"/>
    <to variable="output" part="payload"
      query="/invoice/formattedDate"/>
  </copy>
</assign>

```

7.13 Manipulating Attributes

You can copy to or from something defined as an XML attribute. An at sign (@) in XPath query syntax refers to an attribute instead of a child element.

7.13.1 How to Manipulate Attributes

The code in [Example 7–27](#) fetches and copies the `custId` attribute from this XML data:

Example 7–27 custId Attribute Fetch and Copy Operations

```

<invalidLoanApplication xmlns="http://samples.otn.com">
  <application xmlns = "http://samples.otn.com/XPath/autoloan">
    <customer custId = "111" >
      <name>
        Mike Olive
      </name>
      ...
    </customer>
    ...
  </application>
</invalidLoanApplication>

```

The code in [Example 7–28](#) selects the `custId` attribute of the customer field and assigns it to the variable `custId`:

Example 7–28 custId Attribute Select and Assign Operations

```

<assign>
  <!-- get the custId attribute and assign to variable custId -->
  <copy>
    <from variable="input" part="payload"
      query="/tns:invalidLoanApplication/autoloan:application
        /autoloan:customer/@custId"/>
    <to variable="custId"/>
  </copy>
</assign>

```

The namespace prefixes in this example are not integral to the example.

The WSDL file defines a customer to have a type in which `custId` is defined as an attribute, as shown in [Example 7-29](#):

Example 7-29 *custId Attribute Definition*

```
<complexType name="CustomerProfileType">
  <sequence>
    <element name="name" type="string"/>
    ...
  </sequence>
  <attribute name="custId" type="string"/>
</complexType>
```

7.14 Manipulating XML Data with bpelx Extensions

You can perform various operations on XML data in assign activities. The `bpelx` extension types described in this section provide this functionality.

7.14.1 How to Use `bpelx:append`

Note: The `bpelx:append` extension is not supported with SDO variables and causes an error.

The `bpelx:append` extension in an assign activity enables a BPEL process service component to append the contents of one variable, expression, or XML fragment to another variable's contents. [Example 7-30](#) provides an example.

Example 7-30 *bpelx:append Extension*

```
<bpel:assign>
  <bpelx:append>
    <bpelx:from ... />
    <bpelx:to ... />
  </bpelx:append>
</bpel:assign>
```

The `from-spec` query within `bpelx:append` yields zero or more nodes. The node list is appended as child nodes to the target node specified by the `to-spec` query.

The `to-spec` query must yield one single L-Value element node. Otherwise, a `bpel:selectionFailure` fault is generated. The `to-spec` query cannot refer to a partner link.

[Example 7-31](#) consolidates multiple bills of material into one single bill of material (BOM) by appending multiple `b:parts` for one BOM to `b:parts` of the consolidated BOM.

Example 7-31 *Consolidation of Multiple Bills of Material*

```
<bpel:assign>
  <bpelx:append>
    <from variable="billOfMaterialVar"
      query="/b:bom/b:parts/b:part" />
    <to variable="consolidatedBillOfMaterialVar"
      query="/b:bom/b:parts" />
  </bpelx:append>
</bpel:assign>
```

```

    </bpelx:append>
  </bpel:assign>

```

7.14.2 How to Use bpelx:insertBefore

Note: The `bpelx:insertBefore` extension works with SDO variables, but the target must be the variable attribute into which the copied data must go.

The `bpelx:insertBefore` extension in an assign activity enables a BPEL process service component to insert the contents of one variable, expression, or XML fragment before another variable's contents. [Example 7-32](#) provides an example.

Example 7-32 `bpelx:insertBefore` Extension

```

<bpel:assign>
  <bpelx:insertBefore>
    <bpelx:from ... />
    <bpelx:to ... />
  </bpelx:insertBefore>
</bpel:assign>

```

The `from-spec` query within `bpelx:insertBefore` yields zero or more nodes. The node list is appended as child nodes to the target node specified by the `to-spec` query.

The `to-spec` query of the `insertBefore` operation points to one or more single L-Value nodes. If multiple nodes are returned, the first node is used as the reference node. The reference node must be an element node. The parent of the reference node must also be an element node. Otherwise, a `bpel:selectionFailure` fault is generated. The node list generated by the `from-spec` query selection is inserted before the reference node. The `to-spec` query cannot refer to a partner link.

[Example 7-33](#) shows the syntax before the execution of `<insertBefore>`. The value of `addrVar` is:

Example 7-33 Presyntax Execution

```

<a:usAddress>
  <a:state>CA</a:state>
  <a:zipcode>94065</a:zipcode>
</a:usAddress>

```

[Example 7-34](#) shows the syntax after the execution:

Example 7-34 Postsyntax Execution

```

<bpel:assign>
  <bpelx:insertBefore>
    <bpelx:from>
      <a:city>Redwood Shore</a:city>
    </bpelx:from>
    <bpelx:to "addrVar" query="/a:usAddress/a:state" />
  </bpelx:insertBefore>
</bpel:assign>

```

[Example 7-35](#) shows the value of `addrVar`:

Example 7–35 addrVar Value

```
<a:usAddress>
  <a:city>Redwood Shore</a:city>
  <a:state>CA</a:state>
  <a:zipcode>94065</a:zipcode>
</a:usAddress>
```

7.14.3 How to Use bpelx:insertAfter

Note: The `bpelx:insertAfter` extension works with SDO variables, but the target must be the variable attribute into which the copied data must go.

The `bpelx:insertAfter` extension in an assign activity enables a BPEL process service component to insert the contents of one variable, expression, or XML fragment after another variable's contents. [Example 7–36](#) provides an example.

Example 7–36 bpelx:insertAfter Extension

```
<bpel:assign>
  <bpelx:insertAfter>
    <bpelx:from ... />
    <bpelx:to ... />
  </bpelx:insertAfter>
</bpel:assign>
```

This operation is similar to the functionality described for [Section 7.14.2, "How to Use bpelx:insertBefore,"](#) except for the following:

- If multiple L-Value nodes are returned by the `to-spec` query, the last node is used as the reference node.
- Instead of inserting nodes before the reference node, the source nodes are inserted after the reference node.

This operation can also be considered a macro of `conditional-switch` + (`append` or `insertBefore`).

[Example 7–37](#) shows the syntax before the execution of `<insertAfter>`. The value of `addrVar` is:

Example 7–37 Presyntax Execution

```
<a:usAddress>
  <a:addressLine>500 Oracle Parkway</a:addressLine>
  <a:state>CA</a:state>
  <a:zipcode>94065</a:zipcode>
</a:usAddress>
```

[Example 7–38](#) shows the syntax after the execution:

Example 7–38 Postsyntax Execution

```
<bpel:assign>
  <bpelx:insertAfter>
    <bpelx:from>
      <a:addressLine>Mailstop 1op6</a:addressLine>
    </bpelx:from>
```

```

        <bpelx:to "addrVar" query="/a:usAddress/a:addressLine[1]" />
    </bpelx:insertAfter>
</bpel:assign>

```

[Example 7–39](#) shows the value of `addrVar`:

Example 7–39 `addrVar` Value

```

<a:usAddress>
  <a:addressLine>500 Oracle Parkway</a:addressLine>
  <a:addressLine>Mailstop 1op6</a:addressLine>
  <a:state>CA</a:state>
  <a:zipcode>94065</a:zipcode>
</a:usAddress>

```

The `from-spec` query within `bpelx:insertAfter` yields zero or more nodes. The node list is appended as child nodes to the target node specified by the `to-spec` query.

7.14.4 How to Use `bpelx:remove`

The `bpelx:remove` extension in an `assign` activity enables a BPEL process service component to remove a variable. [Example 7–40](#) provides an example.

Example 7–40 `bpelx:remove` Extension

```

<bpel:assign>
  <bpelx:remove>
    <bpelx:target variable="ncname" part="ncname"? query="xpath_str" />
  </bpelx:append>
</bpel:assign>

```

Node removal specified by the XPath expression is supported. Nodes specified by the XPath expression can be multiple, but must be L-Values. Nodes being removed from this parent can be text nodes, attribute nodes, and element nodes.

The XPath expression can return one or more nodes. If the XPath expression returns zero nodes, then a `bpel:selectionFailure` fault is generated.

The syntax of `bpelx:target` is similar to and a subset of `to-spec` for the `copy` operation.

[Example 7–41](#) shows `addrVar` with the following value:

Example 7–41 `addrVar`

```

<a:usAddress>
  <a:addressLine>500 Oracle Parkway</a:addressLine>
  <a:addressLine>Mailstop 1op6</a:addressLine>
  <a:state>CA</a:state>
  <a:zipcode>94065</a:zipcode>
</a:usAddress>

```

After executing the syntax shown in [Example 7–42](#) in the BPEL process service component file, the second address line of `Mailstop` is removed:

Example 7–42 `Removal of Second Address Line`

```

<bpel:assign>
  <bpelx:remove>
    <target variable="addrVar"

```

```

        query="/a:usAddress/a:addressLine[2]" />
    </bpelx:remove>
</bpel:assign>

```

After executing the syntax shown in [Example 7-43](#) in the BPEL process service component file, both address lines are removed:

Example 7-43 Removal of Both Address Lines

```

<bpel:assign>
  <bpelx:remove>
    <target variable="addrVar"
      query="/a:usAddress/a:addressLine" />
  </bpelx:remove>
</bpel:assign>

```

7.14.5 How to Use bpelx:rename and XSD Type Casting

The `bpelx:rename` extension in an assign activity enables a BPEL process service component to rename an element through use of XSD type casting. [Example 7-44](#) provides an example.

Example 7-44 bpelx:rename Extension

```

<bpel:assign>
  <bpelx:rename elementTo="QName1"? typeCastTo="QName2"?>
    <bpelx:target variable="ncname" part="ncname"? query="xpath_str" />
  </bpelx:rename>
</bpel:assign>

```

The syntax of `bpelx:target` is similar to and a subset of `to-spec` for the copy operation. The target must return a list of one more element nodes. Otherwise, a `bpel:selectionFailure` fault is generated. The element nodes specified in the `from-spec` are renamed to the `QName` specified by the `elementTo` attribute. The `xsi:type` attribute is added to those element nodes to cast those elements to the `QName` type specified by the `typeCastTo` attribute.

Assume you have the employee list shown in [Example 7-45](#):

Example 7-45 xsi:type Attribute

```

<e:empList>
  <e:emp>
    <e:firstName>John</e:firstName><e:lastName>Dole</e:lastName>
  <e:emp>
  <e:emp xsi:type="e:ManagerType">
    <e:firstName>Jane</e:firstName><e:lastName>Dole</e:lastName>
    <e:approvalLimit>3000</e:approvalLimit>
    <e:managing />
  <e:emp>
  <e:emp>
    <e:firstName>Peter</e:firstName><e:lastName>Smith</e:lastName>
  <e:emp>
  <e:emp>
    <e:firstName>Mary</e:firstName><e:lastName>Smith</e:lastName>
  <e:emp>
</e:empList>

```

Promotion changes are now applied to Peter Smith in the employee list in [Example 7-46](#):

Example 7-46 Application of Promotion Changes

```

<bpel:assign>
  <bpelx:rename typeCastTo="e:ManagerType">
    <bpelx:target variable="empListVar"
      query="/e:empList/e:emp[./e:firstName='Peter' and
./e:lastName='Smith' " />
    </bpelx:rename>
  </bpel:assign>

```

After executing the above casting (renaming), the data looks as shown in [Example 7-47](#) with `xsi:type` info added to Peter Smith:

Example 7-47 Data Output

```

<e:empList>
  <e:emp>
    <e:firstName>John</e:firstName><e:lastName>Dole</e:lastName>
  <e:emp>
  <e:emp xsi:type="e:ManagerType">
    <e:firstName>Jane</e:firstName><e:lastName>Dole</e:lastName>
    <e:approvalLimit>3000</e:approvalLimit>
    <e:managing />
  <e:emp>
  <e:emp xsi:type="e:ManagerType">
    <e:firstName>Peter</e:firstName><e:lastName>Smith</e:lastName>
  <e:emp>
  <e:emp>
    <e:firstName>Mary</e:firstName><e:lastName>Smith</e:lastName>
  <e:emp>
</e:empList>

```

The employee data of Peter Smith is now invalid, because `<approvalLimit>` and `<managing>` are missing. Therefore, `<append>` is used to add that information. [Example 7-48](#) provides an example.

Example 7-48 Use of append Extension to Add Information

```

<bpel:assign>
  <bpelx:rename typeCastTo="e:ManagerType">
    <bpelx:target variable="empListVar"
      query="/e:empList/e:emp[./e:firstName='Peter' and
./e:lastName='Smith' " />
    </bpelx:rename>
  <bpelx:append>
    <bpelx:from>
      <e:approvalLimit>2500</e:approvalLimit>
      <e:managing />
    </bpelx:from>
    <bpelx:to variable="empListVar"
      query="/e:empList/e:emp[./e:firstName='Peter' and
./e:lastName='Smith' " />
    </bpelx:append>
  </bpel:assign>

```

With the execution of both `rename` and `append`, the corresponding data looks as shown in [Example 7-49](#):

Example 7-49 rename and append Execution

```

<e:emp xsi:type="e:ManagerType">

```

```

    <e:firstName>Peter</e:firstName><e:lastName>Smith</e:lastName>
    <e:approvalLimit>2500</e:approvalLimit>
    <e:managing />
  </e:emp>

```

7.14.6 How to Use bpelx:copyList

The `bpelx:copyList` extension in an assign activity enables a BPEL process service component to perform a `copyList` operation of the contents of one variable, expression, or XML fragment to another variable. [Example 7–50](#) provides an example.

Example 7–50 `bpelx:copyList` Extension

```

<bpel:assign>
  <bpelx:copyList>
    <bpelx:from ... />
    <bpelx:to ... />
  </bpelx:copyList>
</bpel:assign>

```

The `from-spec` query can yield a list of either all attribute nodes or all element nodes. The `to-spec` query can yield a list of L-value nodes: either all attribute nodes or all element nodes.

All the element nodes returned by the `to-spec` query must have the same parent element. If the `to-spec` query returns a list of element nodes, all element nodes must be contiguous.

If the `from-spec` query returns attribute nodes, then the `to-spec` query must return attribute nodes. Likewise, if the `from-spec` query returns element nodes, then the `to-spec` query must return element nodes. Otherwise, a `bpws:mismatchedAssignmentFailure` fault is thrown.

The `from-spec` query can return zero nodes, while the `to-spec` query must return at least one node. If the `from-spec` query returns zero nodes, the effect of the `copyList` operation is similar to the `remove` operation.

The `copyList` operation provides the following features:

- Removes all the nodes pointed to by the `to-spec` query.
- If the `to-spec` query returns a list of element nodes and there are leftover child nodes after removal of those nodes, the nodes returned by the `from-spec` query are inserted before the next sibling of the last element specified by the `to-spec` query. If there are no leftover child nodes, an `append` operation is performed.
- If the `to-spec` query returns a list of attribute nodes, those attributes are removed from the parent element. The attributes returned by the `from-spec` query are then appended to the parent element.

For example, assume a schema is defined as shown in [Example 7–51](#).

Example 7–51 Schema

```

<schema attributeFormDefault="unqualified"
  elementFormDefault="qualified"
  targetNamespace="http://xmlns.oracle.com/Event_jws/Event/EventTest"
  xmlns="http://www.w3.org/2001/XMLSchema">
  <element name="process">
    <complexType>
      <sequence>
        <element name="payload" type="string"

```

```

                maxOccurs="unbounded" />
            </sequence>
        </complexType>
    </element>
    <element name="processResponse">
        <complexType>
            <sequence>
                <element name="payload" type="string"
                    maxOccurs="unbounded" />
            </sequence>
        </complexType>
    </element>
</schema>

```

The `from` variable contains the content shown in [Example 7-52](#).

Example 7-52 Variable Content

```

<ns1:process xmlns:ns1="http://xmlns.oracle.com/Event_jws/Event/EventTest">
    <ns1: payload >a</ns1: payload >
    <ns1: payload >b</ns1: payload >
</ns1:process>

```

The `to` variable contains the content shown in [Example 7-53](#).

Example 7-53 Variable Content

```

<ns1:processResponse xmlns:ns1="http://xmlns.oracle.com/Event_
    jws/Event/EventTest">
    <ns1: payload >c</ns1: payload >
</ns1:process>

```

The `bpelx:copyList` operation looks as shown in [Example 7-54](#).

Example 7-54 bpelx:copyList

```

<assign>
    <bpelx:copyList>
        <bpelx:from variable="inputVariable" part="payload"
            query="/client:process/client:payload"/>
        <bpelx:to variable="outputVariable" part="payload"
            query="/client:processResponse/client:payload"/>
    </bpelx:copyList>
</assign>

```

This makes the `to` variable as shown in [Example 7-55](#).

Example 7-55 Variable Content

```

<ns1:processResponse xmlns:ns1="http://xmlns.oracle.com/Event_
    jws/Event/EventTest">
    <ns1: payload >a</ns1: payload >
    <ns1: payload >b</ns1: payload >
</ns1:process>

```

7.15 Validating XML Data with bpelx:validate

The `bpelx:validate` function enables you to verify code and identify invalid XML data.

7.15.1 How to Validate XML Data with `bpelx:validate`

Use this extension as follows:

- With the `validate` attribute in an `assign` activity:

```
<assign bpelx:validate="yes|no">
...
</assign>
```

- In `<bpelx:validate>` as a standalone, extended activity that can be used without an `assign` activity:

```
<bpelx:validate variables="NCNAMES" />
```

For example:

```
<bpelx:validate variables="myMsgVariable myPOElemVar" />
```

7.16 Manipulating XML Data Sequences That Resemble Arrays

Data sequences are one of the most basic data models used in XML. However, manipulating them can be nontrivial. One of the most common data sequence patterns used in BPEL process service components are arrays. Based on the XML schema, the way you can identify a data sequence definition is by its attribute `maxOccurs` being set to a value greater than one or marked as unbounded. See the *XML Schema Specification* at <http://www.w3.org/TR> for more information.

The examples in this section illustrate several basic ways of manipulating data sequences in BPEL. However, there are other associated requirements, such as performing looping or dynamic referencing of endpoints. For additional code samples and further information regarding real-world use cases for data sequence manipulation in BPEL, see http://www.oracle.com/technology/sample_code/products/bpel.

The following sections describe a particular requirement for data sequence manipulation.

7.16.1 How to Statically Index into an XML Data Sequence That Uses Arrays

The following two examples illustrate how to use XPath functionality to select a data sequence element when the index of the element you want is known at design time. In these cases, it is the first element.

In [Example 7-56](#), `addresses[1]` selects the first element of the `addresses` data sequence:

Example 7-56 Data Sequence Element Selection

```
<assign>
  <!-- get the first address and assign to variable address -->
  <copy>
    <from variable="input" part="payload"
      query="/tns:invalidLoanApplication/autoloan:application
        /autoloan:customer/autoloan:addresses[1]"/>
    <to variable="address"/>
  </copy>
</assign>
```

In this query, `addresses[1]` is equivalent to `addresses[position()=1]`, where `position` is one of the core XPath functions (see sections 2.4 and 4.1 of the *XML Path*

Language (XPath) Specification). The query in [Example 7-57](#) calls the `position` function explicitly to select the first element of the addresses data sequence. It then selects that address's `street` element (which the activity assigns to the variable `street1`).

Example 7-57 position Function Use

```
<assign>
  <!-- get the first address's street and assign to street1 -->
  <copy>
    <from variable="input" part="payload"
      query="/tns:invalidLoanApplication/autoloan:application
        /autoloan:customer/autoloan:addresses[position()=1]
        /autoloan:street"/>
    <to variable="street1"/>
  </copy>
</assign>
```

If you review the definition of the input variable and its payload part in the WSDL file, you go several levels down before coming to the definition of the addresses field. There you see the `maxOccurs="unbounded"` attribute. The two XPath indexing methods are functionally identical; you can use whichever method you prefer.

7.16.2 How to Determine Sequence Size

If you must know the runtime size of a data sequence (that is, the number of nodes or data items in the sequence), you can get it by using the combination of the XPath built-in `count()` function and the BPEL built-in `getVariableData()` function.

The code in [Example 7-58](#) calculates the number of elements in the `item` sequence and assigns it to the integer variable `lineItemSize`.

Example 7-58 Sequence Size Determination

```
<assign>
  <copy>
    <from expression="count(bpws:getVariableData('outpoint', 'payload',
      '/p:invoice/p:lineItems/p:item'))"/>
    <to variable="lineItemSize"/>
  </copy>
</assign>
```

7.16.3 How to Dynamically Index by Applying a Trailing XPath to an Expression

Often a dynamic value is needed to index into a data sequence; that is, you must get the *n*th node out of a sequence, where the value of *n* is defined at runtime. This section covers the methods for dynamically indexing by applying a trailing XPath into expressions.

7.16.3.1 Applying a Trailing XPath to the Result of getVariableData

The dynamic indexing method shown in [Example 7-59](#) applies a trailing XPath to the result of `bpws:getVariableData()`, instead of using an XPath as the last argument of `bpws:getVariableData()`. The trailing XPath references to an integer-based index variable within the position predicate (that is, `[...]`).

Example 7-59 Dynamic Indexing

```
<variable name="idx" type="xsd:integer"/>
```

```

...
<assign>
  <copy>
    <from expression="bpws:getVariableData('input', 'payload'
      )/p:line-item[bpws:getVariableData('idx')]/p:line-total" />
    <to variable="lineTotalVar" />
  </copy>
</assign>
    
```

Assume at runtime that the `idx` integer variable holds 2 as its value. The preceding expression within the `from` is equivalent to:

```

<from expression="bpws:getVariableData('input', 'payload'
  )/p:line-item[2]/p:line-total" />
    
```

There are some subtle XPath usage differences, when an XPath used trailing behind the `bpws:getVariableData()` function is compared with the one used inside the function.

Using the same example (where `payload` is the message part of element `"p:invoice"`), if the XPath is used within the `getVariableData()` function, the root element name (`" /p:invoice"`) must be specified at the beginning of the XPath.

For example:

```

bpws:getVariableData('input', 'payload',
  '/p:invoice/p:line-item[2]/p:line-total')
    
```

If the XPath is used trailing behind the `bpws:getVariableData()` function, the root element name does not need to be specified in the XPath.

For example:

```

bpws:getVariableData('input', 'payload')/p:line-item[2]/p:line-total
    
```

This is because the node returned by the `getVariableData()` function is the root element. Specifying the root element name again in the XPath is redundant and is incorrect according to standard XPath semantics.

7.16.3.2 Using the `bpelx:append` Extension to Append New Items to a Sequence

The `bpelx:append` extension in an `assign` activity enables BPEL process service components to append new elements to an existing parent element. [Example 7-60](#) provides an example.

Example 7-60 *bpelx:append Extension*

```

<assign name="assign-3">
  <copy>
    <from expression="bpws:getVariableData('idx')+1" />
    <to variable="idx"/>
  </copy>
  <bpelx:append>
    <bpelx:from variable="partInfoResultVar" part="payload" />
    <bpelx:to variable="output" part="payload" />
  </bpelx:append>
  ...
</assign>
    
```

The `bpelx:append` logic in this example appends the `payload` element of the `partInfoResultVar` variable as a child to the `payload` element of the `output`

variable. In other words, the payload element of `output` variable is used as the parent element.

7.16.3.3 Merging Data Sequences

You can merge two sequences into a single data sequence. This pattern is common when the data sequences are in an array (that is, the sequence of data items of compatible types).

The two `append` operations shown in [Example 7–61](#) under `assign` demonstrate how to merge data sequences:

Example 7–61 Data Sequences Merges with `append` Operations

```
<assign>
  <!-- initialize "mergedLineItems" variable
        to an empty element -->
  <copy>
    <from> <p:lineItems /> </from>
    <to variable="mergedLineItems" />
  </copy>
  <bpelx:append>
    <bpelx:from variable="input" part="payload"
      query="/p:invoice/p:lineItems/p:lineitem" />
    <bpelx:to variable="mergedLineItems" />
  </bpelx:append>
  <bpelx:append>
    <bpelx:from variable="literalLineItems"
      query="/p:lineItems/p:lineitem" />
    <bpelx:to variable="mergedLineItems" />
  </bpelx:append>
</assign>
```

7.16.3.4 Generating Functionality Equivalent to an Array of an Empty Element

The `genEmptyElem` function generates functionality equivalent to an array of an empty element to an XML structure. This function takes the following arguments:

```
genEmptyElem('ElemQName',int?, 'TypeQName'?, boolean?)
```

Note the following issues:

- The first argument specifies the `QName` of the empty elements.
- The optional second integer argument specifies the number of empty elements. If missing, the default size is 1.
- The third optional argument specifies the `QName`, which is the `xsi:type` of the generated empty name. This `xsi:type` pattern matches the `SOAPENC:Array`. If it is missing or is an empty string, the `xsi:type` attribute is not generated.
- The fourth optional boolean argument specifies whether the generated empty elements are XSI - `nil`, provided the element is XSD-nillable. The default value is `false`. If missing or `false`, `xsi:nil` is not generated.

[Example 7–62](#) shows an `append` statement initializing a purchase order (PO) document with 10 empty `<lineItem>` elements under `po`:

Example 7–62 `append` Statement

```
<bpelx:assign>
  <bpelx:append>
```

```

    <bpelx:from expression="ora:genEmptyElem('p:lineItem',10)" />
    <bpelx:to variable="poVar" query="/p:po" />
  </bpelx:append>
</bpelx:assign>

```

The `genEmptyElem` function in this example can be replaced with an embedded XQuery expression:

```

ora:genEmptyElem('p:lineItem',10)
== for $i in (1 to 10) return <p:lineItem />

```

The empty elements generated by this function are typically invalid XML data. You perform further data initialization after the empty elements are created. Using the same example above, you can perform the following:

- Add attribute and child elements to those empty `lineItem` elements.
- Perform `copy` operations to replace the empty elements. For example, copy from a web service result to an individual entry in this equivalent array under a `flowN` activity.

7.16.4 What You May Need to Know About SOAP-Encoded Arrays

Oracle BPEL Process Manager provides limited support for Simple Object Access Protocol (SOAP)-encoded arrays (`soapenc:arrayType`).

Consider one of the following methodologies to deal with SOAP arrays:

- Place a wrapper around the service so that the BPEL process service component talks to the document literal wrapper service, which in turn calls the underlying service with `soapenc:arrayType`.
- Call a service with `soapenc:arrayType` from BPEL, but construct the XML message more manually in the BPEL code. This action enables you to avoid changing or wrapping the service. However, each time you want to call that service from BPEL, you must take extra steps.

7.16.5 What You May Need to Know About Using the Array Identifier

For processing in Native Format Builder array identifier environments, information is required about the parent node of a node. Because the `reportSAXEvents` API is used, this information is typically not available for outbound message scenarios. Setting `nxsd:useArrayIdentifiers` to `true` in the native schema enables DOM-parsing to be used for outbound message scenarios. Use this setting cautiously, as it can lead to slower performance for very large payloads.

```

<?xml version="1.0" ?>

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
  targetNamespace="http://xmlns.oracle.com/pcbpel/demoSchema/csv"
  xmlns:tns="http://xmlns.oracle.com/pcbpel/demoSchema/csv"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified" nxsd:encoding="US-ASCII"
nxsd:stream="chars" nxsd:version="NXSD" nxsd:useArrayIdentifiers="true">
  <xsd:element name="Root-Element">
    ....
  </xsd:element>
</xsd:schema>

```

7.17 Converting from a String to an XML Element

Sometimes a service is defined to return a string, but the content of the string is actually XML data. The problem is that, although BPEL provides support for manipulating XML data (using XPath queries, expressions, and so on), this functionality is not available if the variable or field is of type string. With Java, you use DOM functions to convert the string to a structured XML object type. You can use the BPEL XPath function `parseEscapedXML` to do the same thing.

7.17.1 How To Convert from a String to an XML Element

The `parseEscapedXML` function takes XML data, parses it through DOM, and returns structured XML data that can be assigned to a typed BPEL variable.

[Example 7-63](#) provides an example:

Example 7-63 String to XML Element Conversion

```
<!-- execute the XPath extension function
parseEscapedXML('&lt;item&gt;') and assign to a variable
-->
<assign>
  <copy>
    <from expression="ora:parseEscapedXML(
      '&lt;item xmlns=&quot;http://samples.otn.com&quot;
      sku=&quot;006&quot;&gt;
      &lt;description&gt;sun ultra sparc VI server
      &lt;/description&gt;
      &lt;price&gt;1000
      &lt;/price&gt;
      &lt;quantity&gt;2
      &lt;/quantity&gt;
      &lt;lineTotal&gt;2000
      &lt;/lineTotal&gt;
      &lt;/item&gt;')"/>
    <to variable="escapedLineItem"/>
  </copy>
</assign>
```

7.18 Understanding the Differences Between Document-Style and RPC-Style WSDL Files

The examples shown up to this point have been for document-style WSDL files in which a message is defined with an XML schema element, as shown in

[Example 7-64](#):

Example 7-64 XML Schema element Definition

```
<message name="LoanFlowRequestMessage">
  <part name="payload" element="s1:loanApplication"/>
</message>
```

This is in contrast to RPC-style WSDL files, in which the message is defined with an XML schema type, as shown in [Example 7-65](#):

Example 7-65 RPC-Style type Definition

```
<message name="LoanFlowRequestMessage">
  <part name="payload" type="s1:LoanApplicationType"/>
```

```
</message>
```

7.18.1 How To Use RPC-Style Files

This impacts the material in this chapter because there is a difference in how XPath queries are constructed for the two WSDL message styles. For an RPC-style message, the top-level element (and therefore the first node in an XPath query string) is the part name (payload in [Example 7-65](#)). In document-style, the top-level node is the element name (for example, loanApplication).

[Example 7-66](#) and [Example 7-67](#) show what an XPath query string looks like if an application named LoanServices were in RPC style.

Example 7-66 RPC-Style WSDL File

```
<message name="LoanServiceResultMessage">
  <part name="payload" type="s1:LoanOfferType"/>
</message>

<complexType name="LoanOfferType">
  <sequence>
    <element name="providerName" type="string"/>
    <element name="selected" type="boolean"/>
    <element name="approved" type="boolean"/>
    <element name="APR" type="double"/>
  </sequence>
</complexType>
```

Example 7-67 RPC-Style BPEL File

```
<variable name="output"
  messageType="tns:LoanServiceResultMessage"/>
...
<assign>
  <copy>
    <from expression="9.9"/>
    <to variable="output" part="payload" query="/payload/APR"/>
  </copy>
</assign>
```

7.19 Manipulating SOAP Headers in BPEL

BPEL's communication activities (invoke, receive, reply, and onMessage) receive and send messages through specified message variables. These default activities permit one variable to operate in each direction. For example, the invoke activity has `inputVariable` and `outputVariable` attributes. You can specify one variable for each of the two attributes. This is enough if the particular operation involved uses only one payload message in each direction.

However, WSDL supports multiple messages in an operation. In the case of SOAP, multiple messages can be sent along the main payload message as SOAP headers. However, BPEL's default communication activities cannot accommodate the additional header messages.

Oracle BPEL Process Manager solves this problem by extending the default BPEL communication activities with the `bpelx:headerVariable` extension. The extension syntax is as shown in [Example 7-68](#):

Example 7–68 *bpelx:headerVariable Extension*

```

<invoke bpelx:inputHeaderVariable="inHeader1 inHeader2 ..."
  bpelx:outputHeaderVariable="outHeader1 outHeader2 ..."
  .../>

<receive bpelx:headerVariable="inHeader1 inHeader2 ..." .../>
<onMessage bpelx:headerVariable="inHeader1 inHeader2 ..." .../>
<reply bpelx:headerVariable="inHeader1 inHeader2 ..." .../>

```

7.19.1 How to Receive SOAP Headers in BPEL

This section provides an example of how to create BPEL and WSDL files to receive SOAP headers.

To receive SOAP headers in BPEL:

1. Create a WSDL file that declares header messages and the SOAP binding that binds them to the SOAP request. [Example 7–69](#) provides an example.

Example 7–69 *WSDL File Contents*

```

<!-- custom header -->
<message name="CustomHeaderMessage">
  <part name="header1" element="tns:header1"/>
  <part name="header2" element="tns:header2"/>
</message>

<binding name="HeaderServiceBinding" type="tns:HeaderService">
  <soap:binding style="document"
    transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="initiate">
    <soap:operation style="document" soapAction="initiate"/>
    <input>
      <soap:header message="tns:CustomHeaderMessage"
        part="header1" use="literal"/>
      <soap:header message="tns:CustomHeaderMessage"
        part="header2" use="literal"/>
      <soap:body use="literal"/>
    </input>
  </operation>
</binding>

```

2. Create a BPEL source file that declares the header message variables and uses `bpelx:headerVariable` to receive the headers, as shown in [Example 7–70](#).

Example 7–70 *bpelx:headerVariable Use*

```

<variables> <variable name="input"
  messageType="tns:HeaderServiceRequestMessage"/>
  <variable name="event"
    messageType="tns:HeaderServiceEventMessage"/>
  <variable name="output"
    messageType="tns:HeaderServiceResultMessage"/>
  <variable name="customHeader"
    messageType="tns:CustomHeaderMessage"/>
</variables>

<sequence>
  <!-- receive input from requestor -->
  <receive name="receiveInput" partnerLink="client"

```

```

    portType="tns:HeaderService" operation="initiate"
    variable="input"
    bpelx:headerVariable="customHeader"
createInstance="yes" />

```

7.19.2 How to Send SOAP Headers in BPEL

This section provides an example of how to send SOAP headers.

To send SOAP headers in BPEL:

1. Define an SCA reference in the `composite.xml` to refer to the `HeaderService`.
2. Define the custom header variable, manipulate it, and send it using `bpelx:inputHeaderVariable`, as shown in [Example 7-71](#).

Example 7-71 `bpelx:inputHeaderVariable` Use

```

<variables>
  <variable name="input" messageType="tns:HeaderTestRequestMessage" />
  <variable name="output" messageType="tns:HeaderTestResultMessage" />
  <variable name="request" messageType="services:HeaderServiceRequestMessage" />
  <variable name="response" messageType="services:HeaderServiceResultMessage" />
  <variable name="customHeader" messageType="services:CustomHeaderMessage" />
</variables>
...
<!-- initiate the remote process -->
<invoke name="invokeAsyncService"
  partnerLink="HeaderService"
  portType="services:HeaderService"
  bpelx:inputHeaderVariable="customHeader"
  operation="initiate"
  inputVariable="request" />

```

7.20 Using MIME/DIME SOAP Attachments

A BPEL process service component can receive SOAP attachments in an optimized Message Transmission Optimization Mechanism (MTOM) format. However, the BPEL process cannot internally process the attachments. Instead, the attachments are added to the DOM as part of the XML file. Oracle recommends that you avoid using MTOM attachments and instead use Multipurpose Internet Mail Extensions (MIME) and Direct Internet Message Encapsulation (DIME) SOAP attachments.

Invoking a Synchronous Web Service from a BPEL Process

This chapter describes how to invoke a synchronous web service from a BPEL process. This chapter demonstrates how to set up the components necessary to perform a synchronous invocation. This chapter also examines how these components are coded.

This chapter includes the following sections:

- [Section 8.1, "Introduction to Invoking a Synchronous Web Service"](#)
- [Section 8.2, "Invoking a Synchronous Web Service"](#)
- [Section 8.3, "Calling a One-Way Mediator with a Synchronous BPEL Process"](#)

For a simple Hello World sample (`bpel-101-HelloWorld`) that takes an input string, adds a prefix of "Hello " to the string, and returns it, visit the following URL:

http://www.oracle.com/technology/sample_code/products/bpel

8.1 Introduction to Invoking a Synchronous Web Service

Synchronous web services provide an immediate response to an invocation. BPEL can connect to synchronous web services through a partner link, send data, and then receive the reply in the same synchronous invocation.

A synchronous invocation requires the following components:

- Partner link

Defines the location and the role of the web services with which the BPEL process service component connects to perform tasks, and the variables used to carry information between the web service and the BPEL process service component. A partner link is required for each web service that the BPEL process service component calls. You can create partner links in either of two ways:

- In the SOA Composite Editor, when you drag a **Web Service** from the Component Palette into the **Exposed Services** or **External References** swimlane.
- In the Oracle BPEL Designer, when you drag a **Partner Link (Web Service/Adapter)** from the Component Palette into the **Partner Links** swimlane. This method is described in this chapter.

- Invoke activity

Opens a port in the BPEL process service component to send and receive data. It uses this port to retrieve information verifying that the customer has acceptable

credit using the **CreditCardAuthorizationService**. For synchronous callbacks, only one port is needed for both the send and receive functions

Note: You can specify timeout values with the attribute `syncMaxWaitTime` in the `Middleware_Home/domains/domain_name/config/soa-infra/configuration/bpel-config.xml` file or with the System MBean Browser setting of `oracle.as.soainfra.config:type=BPELConfig,name=bpel` in Oracle Enterprise Manager Fusion Middleware Control Console. If the BPEL process service component does not receive a reply within the specified time, then the activity fails.

8.2 Invoking a Synchronous Web Service

This section examines a synchronous invocation operation using the `OrderProcessor.bpel` file in the WebLogic Fusion Order Demo application as an example. For a more step-by-step approach, see *Oracle Fusion Middleware Tutorial for Running and Building an Application with Oracle SOA Suite*.

8.2.1 How to Invoke a Synchronous Web Service

To invoke a synchronous web service:

1. In the Component Palette in Oracle BPEL Designer, drag the necessary partner link, invoke activity, and assign activities into the designer.
2. Edit their dialogs. Procedures are described in *Oracle Fusion Middleware Tutorial for Running and Building an Application with Oracle SOA Suite*.

Figure 8–1 shows the diagram for the **Scope_AuthorizeCreditCard** scope activity of the `OrderProcessor.bpel` file, which defines a simple set of actions.

Figure 8–1 Diagram of OrderProcessor.bpel

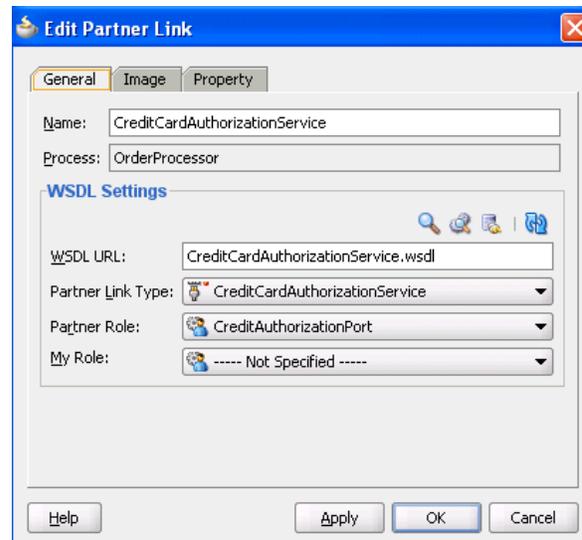


The following actions take place:

1. The **Assign_CreditCheckInput** **assign** activity packages the data from the client. The assign activity provides a method for copying the contents of one variable to another. In this case, it takes the credit card type, credit card number, and purchase amount and assigns them to the input variable for the **CreditAuthorizationService** service.

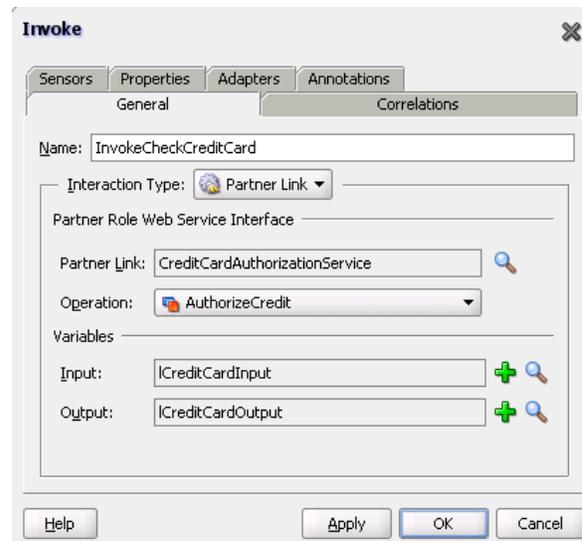
- The **InvokeCheckCreditCard** activity calls the **CreditCardAuthorization** service. [Figure 8–2](#) shows the **CreditCardAuthorizationService** web service, which is defined as a partner link.

Figure 8–2 CreditCardAuthorizationService Partner Link



[Figure 8–3](#) shows the **InvokeCheckCreditCard** invoke activity.

Figure 8–3 InvokeCheckCreditCard Invoke Activity



- The **Switch_EvaluateCCResult** switch activity checks the results of the credit card validation. For information about switch activities, see [Section 11.2, "Creating a Switch Activity to Define Conditional Branching."](#)

8.2.2 What Happens When You Invoke a Synchronous Web Service

When you create a partner link and invoke activity, the necessary BPEL code for invoking a synchronous web service is added to the appropriate BPEL and Web Services Description Language (WSDL) files.

8.2.2.1 Partner Link in the BPEL Code

In the `OrderProcessor.bpel` code, the partner link defines the link name and type, and the role of the BPEL process service component in interacting with the partner service.

From the BPEL source code, the `CreditCardAuthorizationService` partner link definition is shown in [Example 8–1](#):

Example 8–1 Partner Link Definition

```
<partnerLink name="CreditCardAuthorizationService"
  partnerRole="CreditAuthorizationPort"
  partnerLinkType="ns2:CreditCardAuthorizationService"/>
```

Variable definitions that are accessible locally in the `Scope_AuthorizeCreditCard` scope are shown in [Example 8–2](#). The types for these variables are defined in the WSDL for the process itself.

Example 8–2 Variable Definition

```
<variable name="lCreditCardInput"
  messageType="ns2:CreditAuthorizationRequestMessage"/>
<variable name="lCreditCardOutput"
  messageType="ns2:CreditAuthorizationResponseMessage"/>
```

The WSDL file defines the interface to your BPEL process service component: the messages that it accepts and returns, the operations that are supported, and other parameters.

8.2.2.2 Partner Link Type and Port Type in the BPEL Code

The web service's `CreditCardAuthorizationService.wsdl` file contains two sections that enable the web service to work with BPEL process service components:

- `partnerLinkType`:
Defines the following characteristics of the conversation between a BPEL process service component and the credit card authorization web service:
 - The role (operation) played by each
 - The `portType` provided by each for receiving messages within the conversation
- `portType`:
A collection of related operations implemented by a participant in a conversation. A port type defines which information is passed back and forth, the form of that information, and so on. A synchronous invocation requires only one port type that both initiates the synchronous process and calls back the client with the response. An asynchronous callback (one in which the reply is not immediate) requires two port types, one to send the request, and another to receive the reply when it arrives.

In this example, the `portType CreditAuthorizationPort` receives the credit card type, credit card number, and purchase amount, and returns the status results.

[Example 8–3](#) provides an example of `partnerLinkType` and `portType`.

Example 8–3 partnerLinkType and portType Definitions

```
<plnk:partnerLinkType name="CreditCardAuthorizationService">
  <plnk:role name="CreditAuthorizationPort">
    <plnk:portType name="tns:CreditAuthorizationPort"/>
  </plnk:role>
</plnk:partnerLinkType>
```

8.2.2.3 Invoke Activity for Performing a Request

The `invoke` activity includes the `lCreditCardInput` local input variable. The credit card authorization web service uses the `lCreditCardInput` input variable. This variable contains the customer's credit card type, credit card number, and purchase amount. The `lCreditCardOutput` variable returns status results from the `CreditAuthorizationService` service. [Example 8–4](#) provides an example.

Example 8–4 Invoke Activity

```
<invoke name="InvokeCheckCreditCard"
  inputVariable="lCreditCardInput"
  outputVariable="lCreditCardOutput"
  partnerLink="CreditCardAuthorizationService"
  portType="ns2:CreditAuthorizationPort"
  operation="AuthorizeCredit"/>
```

8.2.2.4 Synchronous Invocation in BPEL Code

The BPEL code shown in [Example 8–5](#) performs the synchronous invocation:

Example 8–5 Synchronous Invocation

```
<assign name="Assign_CreditCheckInput">
  <copy>
    <from variable="gOrderInfoVariable"
      query="/ns4:orderInfoVOSDO/ns4:OrderTotal"/>
    <to variable="lCreditCardInput" part="Authorization"
      query="/ns8:AuthInformation/ns8:PurchaseAmount"/>
  </copy>
  <copy>
    <from variable="gOrderInfoVariable"
      query="/ns4:orderInfoVOSDO/ns4:CardTypeCode"/>
    <to variable="lCreditCardInput" part="Authorization"
      query="/ns8:AuthInformation/ns8:CCType"/>
  </copy>
  <copy>
    <from variable="gOrderInfoVariable"
      query="/ns4:orderInfoVOSDO/ns4:AccountNumber"/>
    <to variable="lCreditCardInput" part="Authorization"
      query="/ns8:AuthInformation/ns8:CCNumber"/>
  </copy>
</assign>
<invoke name="InvokeCheckCreditCard"
  inputVariable="lCreditCardInput"
  outputVariable="lCreditCardOutput"
  partnerLink="CreditCardAuthorizationService"
  portType="ns2:CreditAuthorizationPort"
  operation="AuthorizeCredit"/>
```

8.3 Calling a One-Way Mediator with a Synchronous BPEL Process

You can expose a synchronous interface in the front end while using an asynchronous callback in the back end to simulate a synchronous reply. This is the default behavior in BPEL processes with the automatic setting of the `configuration.transaction` property to `requiresNew` in the `composite.xml` file. [Example 8–6](#) provides details.

Example 8–6 *configuration.transaction Property*

```
<component name="BPELProcess1">
@ <implementation.bpel src="BPELProcess1.bpel"/>
@ <property name="configuration.transaction" type="xs:string"
@ many="false">requiresNew</property>
@ </component>
```

`RequiresNew` is the recommended value. If you want to participate in the client's transaction, you must set the `configuration.transaction` property to `Required`.

Invoking an Asynchronous Web Service from a BPEL Process

This chapter describes how to call an asynchronous web service. Asynchronous messaging styles are useful for environments in which a service, such as a loan processor, can take a long time to process a client request. Asynchronous services also provide a more reliable fault-tolerant and scalable architecture than synchronous services.

This chapter includes the following sections:

- [Section 9.1, "Introduction to Invoking an Asynchronous Web Service"](#)
- [Section 9.2, "Invoking an Asynchronous Web Service"](#)
- [Section 9.3, "Using WS-Addressing in an Asynchronous Service"](#)
- [Section 9.4, "Using Correlation Sets in an Asynchronous Service"](#)

9.1 Introduction to Invoking an Asynchronous Web Service

This section introduces asynchronous web service invocation with a company called United Loan. United Loan publishes an asynchronous web service that processes a client's loan application request and then returns a loan offer. This use case discusses how to integrate a BPEL process service component with this asynchronous loan application approver web service.

This use case illustrates the key design concepts for requesting information from an asynchronous service, and then receiving the response. The asynchronous United Loan service in this example is another BPEL process service component. However, the same BPEL call can interact with any properly designed web service. The target web service WSDL file contains the information necessary to request and receive the necessary information.

For the asynchronous web service, the following actions take place (in order of priority):

1. An assign activity prepares the loan application.
2. An invoke activity initiates the loan request. The contents of this request are put into a request variable. This request variable is sent to the asynchronous loan processor web service.

When the loan request is initiated, a correlation ID unique to the client and partner link initiating the request is also sent to the loan processor web service. The correlation ID ensures that the correct loan offer response is returned to the corresponding loan application requester.

3. The loan processor web service then sends the correct response to the receive activity, which has been tracked by the correlation ID.
4. An assign activity reads the loan application offer.

The remaining sections in this chapter provide specific details about the asynchronous functionality.

9.2 Invoking an Asynchronous Web Service

This section provides an overview of the tasks for adding asynchronous functionality to a BPEL process service component.

9.2.1 How to Invoke an Asynchronous Web Service

You perform the following steps to asynchronously invoke a web service:

- Add a partner link
- Add an invoke activity
- Add a receive activity
- Create assign activities

9.2.1.1 Adding a Partner Link for an Asynchronous Service

These instructions describe how to create a partner link in a BPEL process (for this example, named LoanService) for the loan application approver web service.

To add a partner link for an asynchronous service:

1. In the SOA Composite Editor, drag a BPEL process from the **Service Components** section of the Component Palette into the designer.

The Create BPEL Process dialog appears.

2. Follow the instructions in the dialog to create a BPEL process service component.
3. Click **OK** when complete.
4. In the SOA composite application in the SOA Composite Editor, double-click the BPEL process service component (for this example, the component is named **LoanBroker**).

The Oracle BPEL Designer appears.

5. In the Component Palette, expand **BPEL Services**.
6. Drag a **Partner Link (Web Service/Adapter)** into the right **Partner Links** swim lane.

The Create Partner Link dialog appears.

7. Enter the following details to create a partner link and select the loan application approver web service:
 - **Name**
Enter a name for the partner link (for this example, LoanService is entered).
 - **Process**
Displays the BPEL process service component name (for this example, LoanBroker appears).

- **WSDL URL**
Enter the name of the Web Services Description Language (WSDL) file to use. Click the **SOA Resource Lookup** icon above this field to locate the correct WSDL.
 - **Partner Link Type**
Refers to the external service with which the BPEL process service component is to interface. Select from the list (for this example, `LoanService` is selected).
 - **Partner Role**
Refers to the role of the external source, for example, provider. Select from the list (for this example, `LoanServiceProvider` is selected).
 - **My Role**
Refers to the role of the BPEL process service component in this interaction. Select from the list (for this example, `LoanServiceRequester` is selected).
8. Click **OK**.
- A new partner link for the loan application approver web service (United Loan) appears in the swim lane of the designer.

9.2.1.2 Adding an Invoke Activity

Follow these instructions to create an invoke activity and a global input variable named `request`. This activity initiates the asynchronous BPEL process service component activity with the loan application approver web service (United Loan). The loan application approver web service uses the `request` input variable to receive the loan request from the client.

To add an invoke activity:

1. In the Component Palette, expand **BPEL Activities and Components**.
2. From the Component Palette, drag an **invoke** activity to beneath the **receive** activity.
3. Go to the Structure window. Note that while this example describes variable creation from the Structure window, you can also create variables by clicking the **Add** icons to the right of the **Input** and **Output** fields of the Invoke dialog.
4. Right-click **Variables** and select **Expand All Child Nodes**.
5. In the second **Variables** folder in the tree, right-click and select **Create Variable**. The Create Variable dialog box appears.
6. Enter the variable name and select **Message Type** from the options provided:
 - **Simple Type**
This option lets you select an XML schema simple type (for example, string, boolean, and so on).
 - **Message Type**
This option enables you to select a WSDL message file definition of a partner link or of the project WSDL file of the current BPEL process service component (for example, a response message or a request message). You can specify variables associated with message types as input or output variables for invoke, receive, or reply activities.

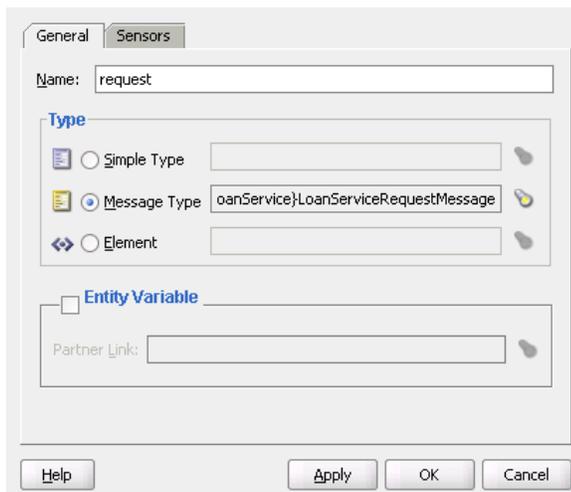
To display the message type, select the **Message Type** option, and then select its **Browse** icon to display the Type Chooser dialog. From here, expand the Message Types tree to make your selection. For this example, **Message Types > Partner Links > Loan Service > LoanService.wsdl > Message Types > LoanServiceRequestMessage** is selected.

- **Element**

This option lets you select an XML schema element of the project schema file or project WSDL file of the current BPEL process service component, or of a partner link.

Figure 9–1 shows the Create Variable dialog.

Figure 9–1 Create Variable Dialog



7. Click **OK**.
8. Double-click the **invoke** activity to display the Invoke dialog.
9. In the Invoke dialog, select the partner link from the **Partner Link** list (for this example, **LoanService** is selected) and **initiate** from the **Operation** list.
10. To the right of the **Input Variable** field, click the second icon and select the input variable you created in Step 6.

The Variable Chooser dialog appears, where you can select the variable.

There is no output variable specified because the output variable is returned in the receive operation. The **invoke** activity is created.

For more information about the invoke activity, see [Section 9.2.2.5, "Invoke and Receive Activities."](#)

11. Click **OK**.

9.2.1.3 Adding a Receive Activity

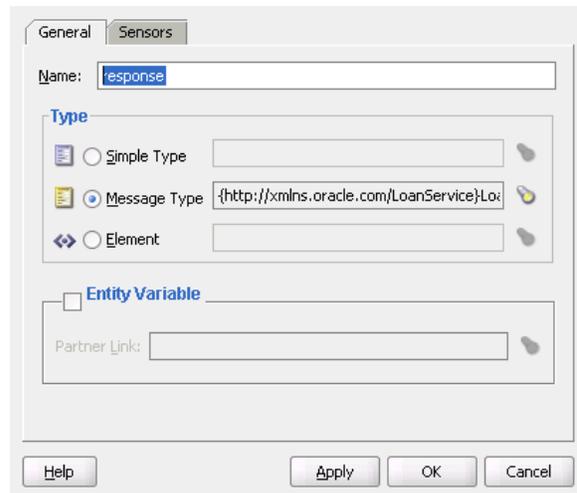
Follow these steps to create a receive activity and a global output variable named `response`. This activity waits for the loan application approver web service's callback operation. The loan application approver web service uses this output variable to send the loan offer result to the client.

To add a receive activity:

1. From the Component Palette, drag a **receive** activity to the location right after the **invoke** activity you created in [Section 9.2.1.2, "Adding an Invoke Activity."](#)
2. Create a variable to hold the receive information by invoking the Create Variable dialog, as you did in Step 3 through Step 7, starting on page 9-3.

[Figure 9-2](#) shows the Create Variable dialog.

Figure 9-2 Create Variable Dialog



3. Double-click the **receive** activity and change its name to `receive_invoke`.
4. From the **Partner Link** list, select the partner link (for this example, **LoanService** is selected).
5. From the **Operation** list, select **onResult**. Do not select the **Create Instance** checkbox.
6. Select the variable you created in Step 3 through Step 7, starting on page 9-3.
7. Click **OK**.

The **receive** activity and the output variable are created. Because the initial **receive** activity in the BPEL file (for this example, **LoanBroker.bpel**) created the initial BPEL process service component instance, a second instance does not need to be created.

9.2.1.4 Performing Additional Activities

In addition to the asynchronous-specific tasks, you must perform the following tasks.

- Create an initial assign activity for data manipulation in front of the invoke activity that copies the client's `input` variable loan application request document payload into the loan application approver web service's `request` variable payload.
- Create a second assign activity for data manipulation after the receive activity that copies the loan application approver web service's `response` variable loan application results payload into the `output` variable for the client to receive.

9.2.2 What Happens When You Invoke an Asynchronous Web Service

This section describes what happens when you invoke an asynchronous web service.

9.2.2.1 portType Section of the WSDL File

The `portType` section of the WSDL file (in this example, for `LoanService`) defines the ports to be used for the asynchronous service.

Asynchronous services have two port types. Each port type performs a one-way operation. In this example, one port type responds to the asynchronous process and the other calls back the client with the asynchronous response. In the example shown in [Example 9-1](#), the `portType` `LoanServiceCallback` receives the client's loan application request and the `portType` `LoanService` asynchronously calls back the client with the loan offer response.

Example 9-1 portType Definition

```
<!-- portType implemented by the LoanService BPEL process -->
  <portType name="LoanService">
    <operation name="initiate">
      <input message="tns:LoanServiceRequestMessage" />
    </operation>
  </portType>
<!-- portType implemented by the requester of LoanService BPEL process
for asynchronous callback purposes
-->
  <portType name="LoanServiceCallback">
    <operation name="onResult">
      <input message="tns:LoanServiceResultMessage" />
    </operation>
  </portType>
```

9.2.2.2 partnerLinkType Section of the WSDL File

The `partnerLinkType` section of the WSDL file (in this example, for `LoanService`) defines the following characteristics of the BPEL process service component:

- The role (operation) played
- The `portType` provided for receiving messages within the conversation

Partner link types in asynchronous services have two roles: one for the web service provider and one for the client requester.

In the conversation shown in [Example 9-2](#), the `LoanServiceProvider` role and `LoanService` `portType` are used for client request messages and the `LoanServiceRequester` role and `LoanServiceCallback` `portType` are used for asynchronously returning (calling back) response messages to the client.

Example 9-2 partnerLinkType Definition

```
<plnk:partnerLinkType name="LoanService">
  <plnk:role name="LoanServiceProvider">
    <plnk:portType name="client:LoanService" />
  </plnk:role>
  <plnk:role name="LoanServiceRequester">
    <plnk:portType name="client:LoanServiceCallback" />
  </plnk:role>
</plnk:partnerLinkType>
```

Two port types are combined into this single asynchronous BPEL process service component: `portType="services:LoanService"` of the `invoke` activity and `portType="services:LoanServiceCallback"` of the `receive` activity. Port types are essentially a collection of operations to be performed. For this BPEL process

service component, there are two operations to perform: `initiate` in the `invoke` activity and `onResult` in the `receive` activity.

9.2.2.3 Partner Links Section in the BPEL File

To call the service from BPEL, you use the BPEL file to define how the process interfaces with the web service. View the `partnerLinks` section. The services with which a process interacts are designed as partner links. Each partner link is characterized by a `partnerLinkType`.

Each partner link is named. This name is used for all service interactions through that partner link. This is critical in correlating responses to different partner links for simultaneous requests of the same type.

Asynchronous processes use a second partner link for the callback to the client. In this example, the second partner link, `LoanService`, is used by the loan application approver web service. [Example 9-3](#) provides an example.

Example 9-3 *partnerLink Definition*

```
<!-- This process invokes the asynchronous LoanService. -->

<partnerLink name="LoanService"
  partnerLinkType="services:LoanService"
  myRole="LoanServiceRequester"
  partnerRole="LoanServiceProvider"/>
</partnerLinks>
```

The attribute `myRole` indicates the role of the client. The attribute `partnerRole` role indicates the role of the partner in this conversation. Each `partnerLinkType` has a `myRole` and `partnerRole` attribute in asynchronous processes.

9.2.2.4 Composite Application File

In the `composite.xml` file, the loan application approver web service appears, as shown in [Example 9-4](#).

Example 9-4 *Loan Application Approver Web Service*

```
<component name="LoanBroker">
  <implementation.bpel process="LoanBroker.bpel"/>
</component>
```

For more information, see [Section 9.2.1.1, "Adding a Partner Link for an Asynchronous Service"](#) for instructions on creating a partner link.

9.2.2.5 Invoke and Receive Activities

View the `variables` and `sequence` sections. Two areas of particular interest concern the `invoke` and `receive` activities:

- An `invoke` activity invokes a synchronous web service (as discussed in [Chapter 8, "Invoking a Synchronous Web Service from a BPEL Process"](#)) or initiates an asynchronous service.

The `invoke` activity includes the `request` global input variable defined in the `variables` section. The `request` global input variable is used by the loan application approver web service. This variable contains the contents of the initial loan application request document.

- A receive activity that waits for the asynchronous callback from the loan application approver web service. The receive activity includes the response global output variable defined in the variables section. This variable contains the loan offer response. The receive activity asynchronously waits for a callback message from a service. While the BPEL process service component is waiting, it is dehydrated, or compressed and stored, until the callback message arrives.

Example 9–5 provides an example.

Example 9–5 Invoke and Receive Activities

```
<variables>

  <variable name="request"
            messageType="services:LoanServiceRequestMessage"/>
  <variable name="response"
            messageType="services:LoanServiceResultMessage"/>
</variables>

<sequence>

  <!-- initialize the input of LoanService -->
  <assign>
  <!-- initiate the remote process -->
  <invoke name="invoke" partnerLink="LoanService"
         portType="services:LoanService"
         operation="initiate" inputVariable="request"/>

  <!-- receive the result of the remote process -->
  <receive name="receive_invoke" partnerLink="LoanService"
          portType="services:LoanServiceCallback"
          operation="onResult" variable="response"/>
```

When an asynchronous service is initiated with the invoke activity, a correlation ID unique to the client request is also sent, using Web Services Addressing (WS-Addressing) (described in [Section 9.3, "Using WS-Addressing in an Asynchronous Service"](#)). Because multiple processes may be waiting for service callbacks, the server must know which BPEL process service component instance is waiting for a callback message from the loan application approver web service. The correlation ID enables the server to correlate the response with the appropriate requesting instance.

9.2.2.6 createInstance Attribute for Starting a New Instance

You may notice a createInstance attribute in the initial receive activity. In this initial receive activity, the createInstance element is set to yes. This starts a new instance of the BPEL process service component. At least one instance startup is required for a conversation. For this reason, you set the createInstance variable to no in the second receive activity.

Example 9–6 shows the source code for the createInstance attribute:

Example 9–6 createInstance Attribute

```
<!-- receive input from requestor -->
<receive name="receiveInput" partnerLink="client"
         portType="tns:LoanBroker"
         operation="initiate" variable="input"
         createInstance="yes"/>
```

9.2.2.7 Dehydration Points for Maintaining Long-Running Asynchronous Processes

To automatically maintain long-running asynchronous processes and their current state information in a database while they wait for asynchronous callbacks, you use a database as a dehydration store. Storing the process in a database preserves the process and prevents any loss of state or reliability if a system shuts down or a network problem occurs. This feature increases both BPEL process service component reliability and scalability. You can also use it to support clustering and failover.

You insert this point between the invoke activity and receive activity.

9.2.2.8 Multiple Runtime Endpoint Locations

Oracle SOA Suite provides support for specifying multiple partner link endpoint locations. This capability is useful for failover purposes if the first endpoint is down. To provide an alternate partner link endpoint location, add the `location` attribute to the `composite.xml` file. [Example 9-7](#) provides an example.

Example 9-7 *Alternate Runtime Endpoint Location*

```
<reference name="HeaderService ...>
<binding.ws port="http://services.otn.com/HelloWorldApp#wsdl.endpoint(client/
  HelloWorldService_pt)"
location="http://server:port/soa-infra/services/default/
  HelloWorldService!1.0/client?WSDL">
<property name="endpointURI">http://jsmith.us.oracle.com:80/a.jsp
@http://myhost.us.oracle.com:8888/soa-infra/services/HelloWorldApp/HelloWorld!
1.0*2007-10-22_14-33-04_195/client
  </property>
</binding.ws>
</reference>
```

9.3 Using WS-Addressing in an Asynchronous Service

Because there can be many active instances at any time, the server must be able to direct web service responses to the correct BPEL process service component instance. You can use WS-Addressing to identify asynchronous messages to ensure that asynchronous callbacks locate the appropriate client.

[Figure 9-3](#) provides an overview of WS-Addressing. WS-Addressing uses Simple Object Access Protocol (SOAP) headers for asynchronous message correlation. Messages are independent of the transport or application used.

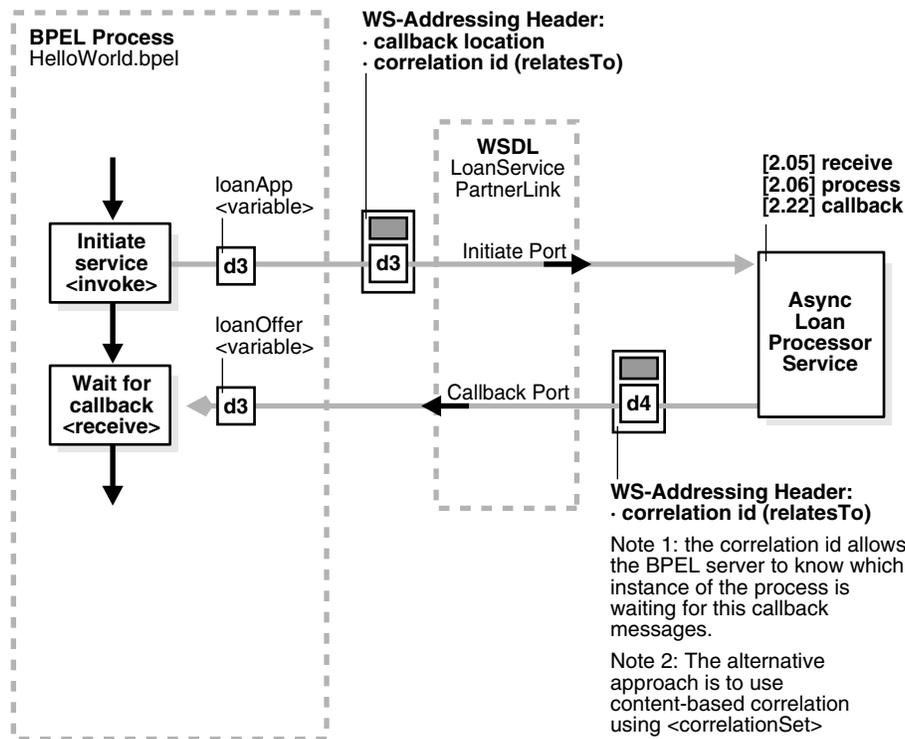
Figure 9–3 Callback with WS-Addressing Headers

Figure 9–3 shows how messages are passed along with WS headers so that the response can be sent to the correct destination.

The example in this chapter uses WS-Addressing for correlation. To view the messages, you can use TCP tunneling, which is described in [Section 9.3.1.1, "Using TCP Tunneling to See Messages Exchanged Between Programs."](#)

WS-Addressing defines the following information typically provided by transport protocols and messaging systems. This information is processed independently of the transport or application:

- Endpoint location (reply-to address)

The reply-to address specifies the location at which a BPEL client is listening for a callback message.

- Conversation ID

Use TCP tunneling to view SOAP messages exchanged between the BPEL process service component flow and the web service (including those containing the correlation ID). You can see the exact SOAP messages that are sent to, or received from, services with which a BPEL process service component flow communicates.

You insert a software listener between your BPEL process service component flow and the web service. Your BPEL process service component flow communicates with the listener (called a TCP tunnel). The listener forwards your messages to the web service, and also displays them. Responses from the web service are returned to the tunnel, which displays and forwards them back to the BPEL process service component.

9.3.1 How to Use WS-Addressing in an Asynchronous Service

WS-Addressing is a public specification and is the default correlation method supported by Oracle BPEL Process Manager. You do not need to edit the `.bpel` and `.wsdl` files to use WS-Addressing.

9.3.1.1 Using TCP Tunneling to See Messages Exchanged Between Programs

The messages that are exchanged between programs and services can be seen through TCP tunneling. This is particularly useful when you want to see the exact SOAP messages exchanged between the BPEL process service component flow and web services.

To monitor the SOAP messages, insert a software listener between your flow and the service. Your flow communicates with the listener (called a TCP tunnel) and the listener forwards your messages to the service, and displays them. Likewise, responses from the service are returned to the tunnel, which displays them and then forwards them back to the flow.

To see all the messages exchanged between the server and a web service, you need only a single TCP tunnel for synchronous services because all the pertinent messages are communicated in a single request and reply interaction with the service. For asynchronous services, you must set up two tunnels, one for the invocation of the service and another for the callback port of the flow.

9.3.1.1.1 Setting up a TCP Listener for Synchronous Services Follow these steps to set up a TCP listener for synchronous services initiated by an Oracle BPEL Process Manager process:

1. Visit the following URL for instructions on how to download and install Axis TCP Monitor (`tcpmon`)

<http://ws.apache.org/commons/tcpmon/>

2. Visit the following URL for instructions on how to use `tcpmon`:

<http://ws.apache.org/axis/java/user-guide.html>

3. Place `axis.jar` in your class path.

4. Start `tcpmon`:

```
C:\...\> java org.apache.axis.utils.tcpmon localport remoteHost
port_on_which_remote_server_is_running
```

5. In the `composite.xml` file, add the `endpointURI` property under `binding.ws` for your flow to override the endpoint of the service.

6. From the operating system command prompt, compile and deploy the process with `ant`.

Note that the same technique can be used to see the SOAP messages passed to invoke a BPEL process service component as a web service from another tool kit such as Axis or .NET.

9.3.1.1.2 Setting up a TCP Listener for Asynchronous Services Follow these steps to set up a TCP listener to display the SOAP messages for callbacks from asynchronous services:

1. Start a TCP listener to listen on a port and to send on the Oracle BPEL Process Manager port.
 - a. Open Oracle Enterprise Manager Fusion Middleware Control Console.

If you are an Oracle JDeveloper user, you can also use the built-in Packet Monitor to see SOAP messages for both synchronous and asynchronous services.

9.4 Using Correlation Sets in an Asynchronous Service

Correlation sets provide another method for directing web service responses to the correct BPEL process service component instance. You can use correlation sets to identify asynchronous messages to ensure that asynchronous callbacks locate the appropriate client.

Correlation sets are a BPEL mechanism that provides for the correlation of asynchronous messages based on message body contents. To use this method, define the correlation sets in your `.bpel` file. This method is designed for services that do not support WS-Addressing or for certain sophisticated conversation patterns, for example, when the conversation is in the form `A > B > C > A` instead of `A > B > A`.

This section describes how to use correlation sets in an asynchronous service with Oracle JDeveloper. Correlation sets enable you to correlate asynchronous messages based on message body contents. You define correlation sets when interactions are not simple invoke-receive activities. This example illustrates how to use correlation sets for a process having three receive activities with no associated invoke activities.

For a sample (`bpel-202-CorrelatedEvents`) that shows how a BPEL process can use correlations for two-way communication using events, visit the following URL:

http://www.oracle.com/technology/sample_code/products/bpel

9.4.1 How to Use Correlation Sets in an Asynchronous Service

This section describes the steps to perform to use correlation sets in an asynchronous service.

9.4.1.1 Step 1: Creating a Project

To create a project:

1. Start Oracle JDeveloper.
2. From the **File** main menu, select **New > Applications**.
3. Select **SOA Application**, and click **OK**.
The Create SOA Application Wizard appears.
4. In the **Application Name** field, enter `MyCorrelationSetApp`.
5. Accept the default values for all remaining settings, and click **Next**.
6. In the **Project Name** field, enter `MyCorrelationSetComposite`.
7. Accept the default values for all remaining settings, and click **Next**.
8. In the **Composite Template** section, select **Composite With BPEL**, and click **Finish**.
The Create BPEL Process dialog appears.
9. Enter the following values:

Table 9–1 Create BPEL Process Dialog Fields and Values

Field	Value
Name	Enter MyCorrelationSet.
Template	Select Asynchronous BPEL Process .
Expose as a SOAP Service	Select the checkbox. After process creation, note the SOAP service that appears in the Exposed Services swimlane. This service provides the entry point to the composite application from the outside world.

10. Accept the default values for all remaining settings, and click **Finish**.

9.4.1.2 Step 2: Configuring Partner Links and File Adapter Services

You now create three partner links that use the SOAP service.

This section contains these topics:

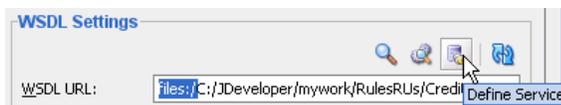
- You create an initial partner link with an adapter service for reading a loan application.
- You create a second partner link with an adapter service for reading an application response.
- You create a third partner link with an adapter service for reading a customer response.

9.4.1.2.1 Creating an Initial Partner Link and File Adapter Service

To create an initial partner link and file adapter service:

1. Double-click the **MyCorrelationSet** BPEL process.
2. In the Component Palette, expand **BPEL Services**.
3. Drag an initial **Partner Link** activity into the right swim lane of the designer.
4. Click the third icon at the top (the **Define Service** icon). This starts the Adapter Configuration Wizard, as shown in [Figure 9–4](#).

Figure 9–4 Adapter Configuration Wizard Startup



5. In the Configure Service or Adapter dialog, select **File Adapter** and click **Next**.
6. In the Welcome dialog, click **Next**.
7. In the **Service Name** field of the Service Name dialog, enter `FirstReceive` and click **Next**.
8. In the Operation dialog, select **Read File** as the **Operation Type** and click **Next**. The **Operation Name** field is automatically filled in with **Read**.
9. Select **Directory Names are Specified as Physical Path**.
10. Above the **Directory for Incoming Files (physical path)** field, click **Browse**.
11. Select a directory from which to read files (for this example, `C:\files\receiveprocess\FirstInputDir` is selected).

12. Click **Select**.
13. Click **Next**.
14. In the File Filtering dialog, enter appropriate file filtering parameters.
15. Click **Next**.
16. In the File Polling dialog, enter appropriate file polling parameters.
17. Click **Next**.
18. In the Messages dialog, click **Browse** next to the **Schema Location** field to display the Type Chooser dialog.
19. Select an appropriate XSD schema file. For this example, **Book1_4.xsd** is the schema and **LoanAppl** is the schema element selected.
20. Click **OK**.

The **Schema Location** field (**Book1_4.xsd** for this example) and the **Schema Element** field (**LoanAppl** for this example) are filled in.

21. Click **Next**.
22. Click **Finish**.

You are returned to the Partner Link dialog. All other fields are automatically completed. The dialog looks as shown in [Table 9-2](#):

Table 9-2 Partner Link Dialog Fields and Values

Field	Value
Name	FirstReceive
WSDL URL	file:/C:/JDeveloper/mywork/Application_Name/SOA_Project_Name/FirstReceive.wsdl where C:/JDeveloper represents the Oracle JDeveloper home directory for this example.
Partner Link Type	Read_plt
Partner Role	Leave unspecified.
My Role	Read_role

23. Click **OK**.

9.4.1.2.2 Creating a Second Partner Link and File Adapter Service

To create a second partner link and file adapter service:

1. Drag a second **PartnerLink** activity beneath the **FirstReceivePL** partner link activity.
2. At the top, click the third icon (the **Define Service** icon).
3. In the Welcome dialog, click **Next**.
4. In the Adapter Type dialog, select **File Adapter** and click **Next**.
5. In the **Service Name** field of the Service Name dialog, enter `SecondFileRead` and click **Next**. This name must be unique from the one you entered in Step 7 on page 9-14.
6. In the Operation dialog, select **Read File** as the **Operation Type**.

7. In the **Operation Name** field, change the name to `Read1`.
8. Click **Next**.
9. Select **Directory Names are Specified as Physical Path**.
10. Above the **Directory for Incoming Files (physical path)** field, click **Browse**.
11. Select a directory from which to read files (for this example, `C:\files\receiveprocess\SecondInputDir` is entered).
12. Click **Select**.
13. Click **Next**.
14. Enter appropriate file filtering parameters in the File Filtering dialog.
15. Click **Next**.
16. Enter appropriate file polling parameters in the File Polling dialog.
17. Click **Next**.
18. Next to the **Schema Location** field in the Messages dialog, click **Browse** to display the Type Chooser dialog.
19. Select an appropriate XSD schema file. For this example, `Book1_5.xsd` is the schema and `LoanAppResponse` is the schema element selected.
20. Click **OK**.
The **Schema Location** field (`Book1_5.xsd` for this example) and the **Schema Element** field (`LoanAppResponse` for this example) are filled in.
21. Click **Next**.
22. Click **Finish**.
You are returned to the Partner Link dialog. All other fields are automatically completed. The dialog looks as shown in [Table 9-3](#):

Table 9-3 Partner Link Dialog Fields and Values

Field	Value
Name	SecondReceive
WSDL URL	<code>file:/C:/JDeveloper/mywork/Application_Name/SOA_Project_Name/SecondFileRead.wsdl</code> where <code>C:/JDeveloper</code> represents the Oracle JDeveloper home directory for this example.
Partner Link Type	Read1_plt
Partner Role	Leave unspecified.
My Role	Read1_role

23. Click **OK**.

9.4.1.2.3 Creating a Third Partner Link and File Adapter Service

To create a third partner link and file adapter service:

1. Drag a third **PartnerLink** activity beneath the **SecondReceivePL** partner link activity.
2. At the top, click the third icon (the **Define Service** icon).

3. In the Welcome dialog, click **Next**.
4. In the Adapter Type dialog, select **File Adapter** and click **Next**.
5. In the **Service Name** field of the Service Name dialog, enter `ThirdFileRead` and click **Next**. This name must be unique from the one you entered in Step 7 on page 9-14 and Step 5 on page 9-15.
6. In the Operation dialog, select **Read File** as the **Operation Type**.
7. In the **Operation Name** field, change the name to `Read2`. This name must be unique.
8. Click **Next**.
9. Select **Directory Names are Specified as Physical Path**.
10. Above the **Directory for Incoming Files (physical path)** field, click **Browse**.
11. Select a directory from which to read files (for this example, `C:\files\receiveprocess\ThirdInputDir` is entered).
12. Click **Select**.
13. Click **Next**.
14. Enter appropriate file filtering parameters in the File Filtering dialog.
15. Click **Next**.
16. Enter appropriate file polling parameters in the File Polling dialog.
17. Click **Next**.
18. Next to the **Schema Location** field in the Messages dialog, click **Browse** to display the Type Chooser dialog.
19. Select an appropriate XSD schema file. For this example, `Book1_6.xsd` is the schema and `CustResponse` is the schema element selected.
20. Click **OK**.
The **Schema Location** field (`Book1_6.xsd` for this example) and the **Schema Element** field (`CustResponse` for this example) are filled in.
21. Click **Next**.
22. Click **Finish**.

You are returned to the Partner Link dialog. All other fields are automatically completed. The dialog looks as shown in [Table 9-4](#):

Table 9-4 Partner Link Dialog Fields and Values

Field	Value
Name	ThirdReceive
WSDL URL	<code>file:/C:/JDeveloper/mywork/Application_Name/SOA_Project_Name/ThirdFileRead.wsdl</code> where <code>C:/JDeveloper</code> represents the Oracle JDeveloper home directory for this example.
Partner Link Type	Read2_plt
Partner Role	Leave unspecified.
My Role	Read2_role

23. Click **OK**.

When complete, the designer looks as shown in [Figure 9–5](#):

Figure 9–5 BPEL Process Design



9.4.1.3 Step 3: Creating Three Receive Activities

You now create three receive activities; one for each partner link. The receive activities specify the partner link from which to receive information.

9.4.1.3.1 Creating an Initial Receive Activity

To create an initial receive activity:

1. Expand **BPEL Activities** in the Component Palette.
2. From the **BPEL Activities and Components** list of the Component Palette section, drag a **Receive** activity beneath the **receiveInput** receive activity in the designer.
3. Double-click the **receive** icon to display the Receive dialog.
4. Enter the details described in [Table 9–5](#) to associate the first partner link (**FirstReceive**) with the first receive activity:

Table 9–5 Receive Dialog Fields and Values

Field	Value
Name	receiveFirst
Partner Link	FirstReceive
Create Instance	Select this checkbox.

The **Operation (Read)** field is automatically filled in.

5. To the right of the **Variable** field, click the first icon. This is the automatic variable creation icon.
6. In the Create Variable dialog, click **OK**.
A variable named **receiveFirst_Read_InputVariable** is automatically created in the **Variable** field.
7. Ensure that you selected the **Create Instance** checkbox, as mentioned in Step 4.
8. Click **OK**.

9.4.1.3.2 Creating a Second Receive Activity

To create a second receive activity:

1. From the Component Palette, drag a second **Receive** activity beneath the **receiveFirst** receive activity.
2. Double-click the **receive** icon to display the Receive dialog.
3. Enter the details described in [Table 9–6](#) to associate the second partner link (**SecondReceivePL**) with the second receive activity:

Table 9–6 *Receive Dialog Fields and Values*

Field	Value
Name	receiveSecond
Partner Link	SecondFileRead
Create Instance	Do <i>not</i> select this checkbox.

The **Operation (Read1)** field is automatically filled in.

4. To the right of the **Variable** field, click the first icon.
5. In the Create Variable dialog, click **OK**.

A variable named **receiveSecond_Read1_InputVariable** is automatically created in the **Variable** field.

6. Click **OK**.

9.4.1.3.3 Creating a Third Receive Activity

To create a third receive activity:

1. From the Component Palette, drag a third **Receive** activity beneath the **receiveSecond** receive activity.
2. Double-click the **receive** icon to display the Receive dialog.
3. Enter the details described in [Table 9–7](#) to associate the third partner link (**ThirdReceivePL**) with the third receive activity:

Table 9–7 *Receive Dialog Fields and Values*

Field	Value
Name	receiveThird
Partner Link	ThirdFileRead
Create Instance	Do <i>not</i> select this checkbox.

The **Operation (Read2)** field is automatically filled in.

4. To the right of the **Variable** field, click the first icon.
5. In the Create Variable dialog, click **OK**.

A variable named **receiveThird_Read2_InputVariable** is automatically created in the **Variable** field.

6. Click **OK**.

Each receive activity is now associated with a specific partner link.

9.4.1.4 Step 4: Creating Correlation Sets

You now create correlation sets. A set of correlation tokens is a set of properties shared by all messages in the correlated group.

9.4.1.4.1 Creating an Initial Correlation Set

To create an initial correlation set:

1. In the Structure window of Oracle JDeveloper, right-click **Correlation Sets** and select **Expand All Child Nodes**.
2. In the second **Correlation Sets** folder, right-click and select **Create Correlation Set**.
3. In the **Name** field of the Create Correlation Set dialog, enter `CorrelationSet1`.
4. In the **Properties** section, click the **Add** icon to display the Property Chooser dialog.
5. Select **Properties**, then click the **Add** icon (first icon at the top) to display the Create Correlation Set Property dialog.
6. In the **Name** field, enter `NameCorr`.
7. To the right of the **Type** field, click the **Browse** icon.
8. In the Type Chooser dialog, select **string** and click **OK**.
9. Click **OK** to close the Create Correlation Set Property dialog, the Property Chooser dialog, and the Create Correlation Set dialog.

9.4.1.4.2 Creating a Second Correlation Set

To create a second correlation set:

1. Return to the **Correlation Sets** section in the Structure window of Oracle JDeveloper.
2. Right-click the **Correlation Sets** folder and select **Create Correlation Set**.
3. In the **Name** field of the Create Correlation Set dialog, enter `CorrelationSet2`.
4. In the **Properties** section, click the **Add** icon to display the Property Chooser dialog.
5. Select **Properties**, then click the **Add** icon to display the Create Correlation Set Property dialog.
6. In the **Name** field, enter `IDCorr`.
7. To the right of the **Type** field, click the **Browse** icon.
8. In the Type Chooser dialog, select **double** and click **OK**.
9. Click **OK** to close the Create Correlation Set Property dialog, the Property Chooser dialog, and the Create Correlation Set dialog.

9.4.1.5 Step 5: Associating Correlation Sets with Receive Activities

You now associate the correlation sets with the receive activities. You perform the following correlation set tasks:

- For the first correlated group, the first and second receive activities are correlated with the **CorrelationSet1** correlation set.
- For the second correlated group, the second and third receive activities are correlated with the **CorrelationSet2** correlation set.

9.4.1.5.1 Associating the First Correlation Set with a Receive Activity

To associate the first correlation set with a receive activity:

1. Double-click the **receiveFirst** receive activity to display the Receive dialog.
2. Click the **Correlations** tab.
3. Click the second **Add** icon to display the Correlation Set Chooser dialog.
4. Select **CorrelationSet1**, then click **OK**.
5. Set the **Initiate** column to **yes**. When set to **yes**, the set is initiated with the values of the properties occurring in the message being exchanged.
6. Click **OK**.

9.4.1.5.2 Associating the Second Correlation Set with a Receive Activity

To associate the second correlation set with a receive activity:

1. Double-click the **receiveSecond** receive activity to display the Receive dialog.
2. Click the **Correlations** tab.
3. Click the second **Add** icon to display the Correlation Set Chooser dialog.
4. Select **CorrelationSet2**, then click **OK**.
5. Set the **Initiate** column to **yes**.
6. Click **Add** and select **CorrelationSet1**.
7. Click **OK**.
8. Set the **Initiate** column to **no** for **CorrelationSet1**.
9. Click **OK**.

This groups the first and second receive activities into a correlated group.

9.4.1.5.3 Associating the Third Correlation Set with a Receive Activity

To associate the third correlation set with a receive activity:

1. Double-click the **receiveThird** receive activity to display the Receive dialog.
2. Click the **Correlations** tab.
3. Click the second **Add** icon to display the Correlation Set Chooser dialog.
4. Select **CorrelationSet2**, then click **OK**.
5. Set the **Initiate** column to **no** for **CorrelationSet2**.
6. Click **OK**.

This groups the second and third receive activities into a second correlated group.

9.4.1.6 Step 6: Creating Property Aliases

Property aliases enable you to map a global property to a field in a specific message part. This action enables the property name to become an alias for the message part and location. The alias can be used in XPath expressions.

9.4.1.6.1 Creating Property Aliases for NameCorr You create the following two property aliases for the **NameCorr** correlation set:

- Map **NameCorr** to the **LoanAppl** message type part of the **receiveFirst** receive activity. This receive activity is associated with the **FirstReceivePL** partner link (defined by the **FirstReceive.wsdl** file).
- Map **NameCorr** to the incoming **LoanAppResponse** message type part of the **receiveSecond** receive activity. This receive activity is associated with the **SecondReceivePL** partner link (defined by the **SecondFileRead.wsdl** file).

To create property aliases for NameCorr:

1. In the Structure window of Oracle JDeveloper, right-click **Property Aliases**.
2. Select **Create Property Alias**.
3. From the **Property** list, select **NameCorr**.
4. Expand and select **Message Types > Web Services > FirstReceivePL > FirstReceive.wsdl > Message Types > LoanAppl_msg > Part - LoanAppl**.
5. In the **Query** field, press Ctrl+Space to define the following XPath expression:


```
/ns2:LoanAppl/ns2:Name
```
6. Click **OK**.
7. Repeat Step 1 through Step 3 to create a second property alias for **NameCorr**.
8. Expand and select **Message Types > Project WSDL Files > SecondFileRead.wsdl > Message Types > LoanAppResponse_msg > Part - LoanAppResponse**.
9. In the **Query** field, press Ctrl+Space to define the following XPath expression:


```
/ns4:LoanAppResponse/ns4:APR
```

9.4.1.6.2 Creating Property Aliases for IDCorr

You create the following two property aliases for the **IDCorr** correlation set:

- Map **IDCorr** to the **LoanAppResponse** message type part of the **receiveSecond** receive activity. This receive activity is associated with the **SecondReceivePL** partner link (defined by the **SecondFileRead.wsdl** file).
- Map **IDCorr** to the **CustResponse** message type part of the **receiveThird** receive activity. This receive activity is associated with the **ThirdReceivePL** partner link (defined by the **ThirdFileRead.wsdl** file).

To create property aliases for IDCorr:

1. In the Structure window, right-click **Property Aliases**.
2. Select **Create Property Alias**.
3. In the **Property** list, select **IDCorr**.
4. Expand and select **Message Types > Project WSDL Files > SecondFileRead.wsdl > Message Types > LoanAppResponse_msg > Part - LoanAppResponse**.
5. In the **Query** field, press Ctrl+Space to define the following XPath expression:


```
/ns4:LoanAppResponse/ns4:APR
```
6. Click **OK**.
7. Repeat Step 1 through Step 3 to create a second property alias for **IDCorr**.
8. Expand and select **Message Types > Project WSDL Files > ThirdFileRead.wsdl > Message Types > CustResponse_msg > Part - CustResponse**.

9. In the **Query** field, press Ctrl+Space to define the following XPath expression:

```
/ns6:CustResponse/ns6:APR
```

Design is now complete.

9.4.1.7 Step 7: Reviewing WSDL File Content

To review WSDL file content:

1. Refresh the Application Navigator.

The NameCorr and IDCorr correlation set properties are defined in the MyCorrelationSet_Properties.wsdl file in the Application Navigator of Oracle JDeveloper. [Example 9–8](#) provides an example.

Example 9–8 Correlation Set Properties

```
<definitions
  name="properties"
  targetNamespace="http://xmlns.oracle.com/MyCorrelationSet/correlationset"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:bpws="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
  xmlns:plnk="http://schemas.xmlsoap.org/ws/2003/05/partner-link/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <bpws:property name="NameCorr" type="xsd:string"/>
  <bpws:property name="IDCorr" type="xsd:double"/>
</definitions>
```

The property aliases are defined in the MyCorrelationSet.wsdl file, as shown in [Example 9–9](#):

Example 9–9 Property Aliases

```
<bpws:propertyAlias propertyName="ns1:NameCorr"
  messageType="ns3:LoanAppl_msg"
  part="LoanAppl" query="/ns2:LoanAppl/ns2:Name" />

<bpws:propertyAlias propertyName="ns1:NameCorr"
  messageType="ns5:LoanAppResponse_msg"
  part="LoanAppResponse" query="/ns4:LoanAppResponse/ns4:APR" />

<bpws:propertyAlias propertyName="ns1:IDCorr"
  messageType="ns5:LoanAppResponse_msg"
  part="LoanAppResponse" query="/ns4:LoanAppResponse/ns4:APR" />

<bpws:propertyAlias propertyName="ns1:IDCorr"
  messageType="ns7:CustResponse_msg"
  part="CustResponse" query="/ns6:CustResponse/ns6:APR" />
```

Because the BPEL process service component is not created as a web services provider in this example, the MyCorrelationSet.wsdl file is not referenced in the BPEL process service component. Therefore, you must import the MyCorrelationSet.wsdl file inside the FirstReceive.wsdl file to reference the correlation sets defined in the former WSDL. [Example 9–10](#) provides an example.

Example 9–10 WSDL File Import

```
<import namespace="http://xmlns.oracle.com/MyCorrelationSet"
```

```
location="MyCorrelationSet.wsdl"/>
```

Using Parallel Flow in a BPEL Process

This chapter describes how to use parallel flow in a BPEL process service component. Parallel flows enable a BPEL process service component to perform multiple tasks at the same time. Parallel flow is especially useful when you must perform several time-consuming and independent tasks.

This chapter includes the following sections:

- [Section 10.1, "Introduction to Parallel Flows in BPEL Processes"](#)
- [Section 10.2, "Creating a Parallel Flow"](#)
- [Section 10.3, "Customizing the Number of Flow Activities with the flowN Activity"](#)

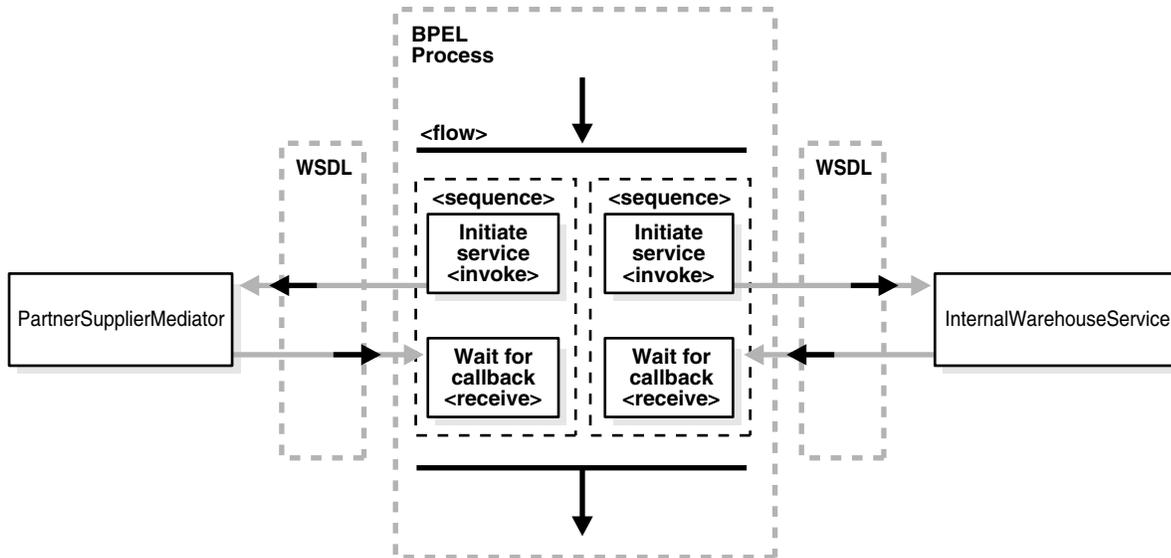
For additional information on creating parallel flows in a SOA composite application, see *Oracle Fusion Middleware Tutorial for Running and Building an Application with Oracle SOA Suite*.

10.1 Introduction to Parallel Flows in BPEL Processes

A BPEL process service component must sometimes gather information from multiple asynchronous sources. Because each callback can take an undefined amount of time (hours or days), it may take too long to call each service one at a time. By breaking the calls into a parallel flow, a BPEL process service component can invoke multiple web services at the same time, and receive the responses as they come in. This method is much more time efficient.

[Figure 10-1](#) shows the **Retrieve_QuotesFromSuppliers** flow activity of the WebLogic Fusion Order Demo application. The **Retrieve_QuotesFromSuppliers** flow activity sends order information to two suppliers in parallel: an internal warehouse (**InternalWarehouseService**) and an external partner warehouse (**PartnerSupplierMediator**). The two warehouses return their bids for the order to the flow activity. Here, two asynchronous callbacks execute in parallel. One callback does not have to wait for the other to complete first. Each response is stored in a different global variable.

Figure 10–1 Parallel Flow Invocation



10.2 Creating a Parallel Flow

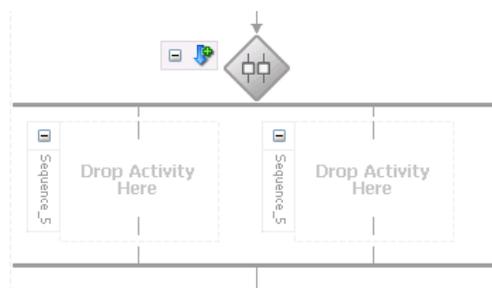
You can create a parallel flow in a BPEL process service component with the flow activity. The flow activity enables you to specify one or more activities to be performed concurrently. The flow activity also provides synchronization. The flow activity completes when all activities in the flow have finished processing. Completion of this activity includes the possibility that it can be skipped if its enabling condition is false.

10.2.1 How to Create a Parallel Flow

To create a parallel flow:

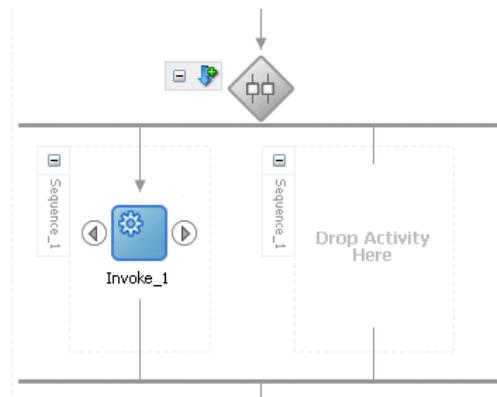
1. From the Component Palette, drag a **Flow** activity into the designer.
2. Click the + sign to expand the flow activity, as shown in [Figure 10–2](#).

Figure 10–2 Flow Activity

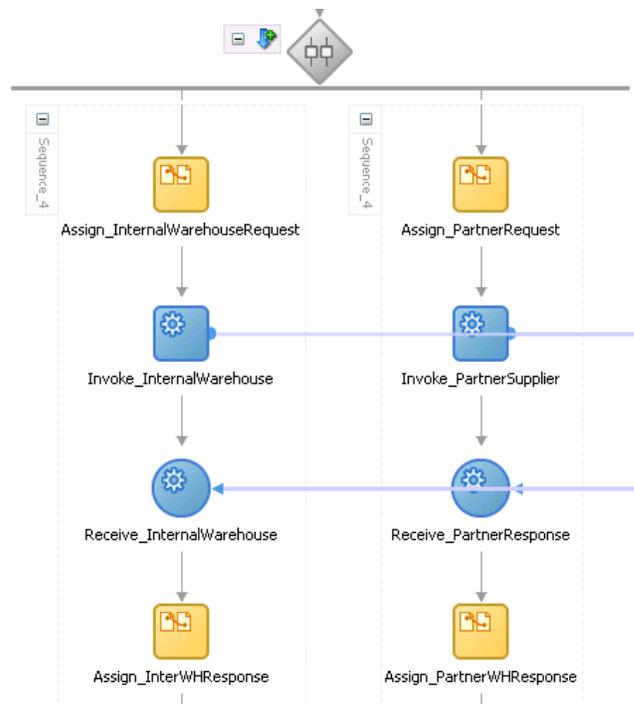


The flow activity includes two branches, each with a box for functional elements. Populate these boxes as you do a scope activity, either by building a function or dragging activities into the boxes.

3. Drag and define additional activities onto each side of the flow to invoke multiple services at the same time.

Figure 10–3 Expanded Flow Activity

When complete, flow activity design can look as shown in [Figure 10–4](#). This example shows the **Retrieve_QuotesFromSuppliers** flow activity of the **WebLogicFusionOrderDemo** application. Two branches are defined for receiving bids, one for **InternalWarehouseService** and the other for **PartnerSupplierMediator**.

Figure 10–4 Flow Activity After Design Completion

10.2.2 What Happens When You Create a Parallel Flow

A flow activity typically contains many sequence activities. Each sequence is performed in parallel. [Example 10–1](#) shows the syntax for two sequences of the **Retrieve_QuotesFromSuppliers** flow activity in the `OrderProcessor.bpel` file after design completion. However, a flow activity can have many sequences. A flow activity can also contain other activities. In [Example 10–1](#), each sequence in the flow contains assign, invoke, and receive activities.

Example 10–1 Flow Activity

```

<flow name="Retrieve_QuotesFromSuppliers">
  <sequence name="Sequence_4">
    <assign name="Assign_InternalWarehouseRequest">
      <copy>
        <from variable="gOrderInfoVariable"
          query="/ns4:orderInfoVOSDO/ns4:OrderId"/>
        <to variable="lInternalWarehouseInputVariable"
          part="payload"
          query="/ns1:WarehouseRequest/ns1:orderId"/>
      </copy>
    </assign>
    <invoke name="Invoke_InternalWarehouse"
      inputVariable="lInternalWarehouseInputVariable"
      partnerLink="InternalWarehouseService"
      portType="ns1:InternalWarehouseService"
      operation="process"/>
    <receive name="Receive_InternalWarehouse"
      createInstance="no"
      variable="lInternalWarehouseResponseVariable"
      partnerLink="InternalWarehouseService"
      portType="ns1:InternalWarehouseServiceCallback"
      operation="processResponse"/>
    <assign name="Assign_InterWHResponse">
      <bpelx:append>
        <bpelx:from variable="lInternalWarehouseResponseVariable"
          part="payload"
          query="/ns1:WarehouseResponse"/>
        <bpelx:to variable="gWarehouseQuotes"
          query="/ns1:WarehouseList"/>
      </bpelx:append>
    </assign>
  </sequence>
  <sequence name="Sequence_4">
    <assign name="Assign_PartnerRequest">
      <copy>
        <from variable="gOrderInfoVariable"
          query="/ns4:orderInfoVOSDO"/>
        <to variable="lPartnerSupplierInputVariable"
          part="request" query="/ns4:orderInfoVOSDO"/>
      </copy>
    </assign>
    <invoke name="Invoke_PartnerSupplier"
      partnerLink="PartnerSupplierMediator"
      portType="ns15:execute_ptt" operation="execute"
      inputVariable="lPartnerSupplierInputVariable"/>
    <receive name="Receive_PartnerResponse"
      createInstance="no"
      variable="lPartnerResponseVariable"
      partnerLink="PartnerSupplierMediator"
      portType="ns15:callback_ptt" operation="callback"/>
    <assign name="Assign_PartnerWHResponse">
      <bpelx:append>
        <bpelx:from variable="lPartnerResponseVariable"
          part="callback"
          query="/ns1:WarehouseResponse"/>
        <bpelx:to variable="gWarehouseQuotes"
          query="/ns1:WarehouseList"/>
      </bpelx:append>
    </assign>
  </sequence>
</flow>

```

```
</sequence>  
</flow>
```

10.3 Customizing the Number of Flow Activities with the flowN Activity

In the flow activity, the BPEL code determines the number of parallel branches. However, often the number of branches required is different depending on the available information. The flowN activity creates multiple flows equal to the value of N, which is defined at runtime based on the data available and logic within the process. An index variable increments each time a new branch is created, until the index variable reaches the value of N.

The flowN activity performs activities on an arbitrary number of data elements. As the number of elements changes, the BPEL process service component adjusts accordingly.

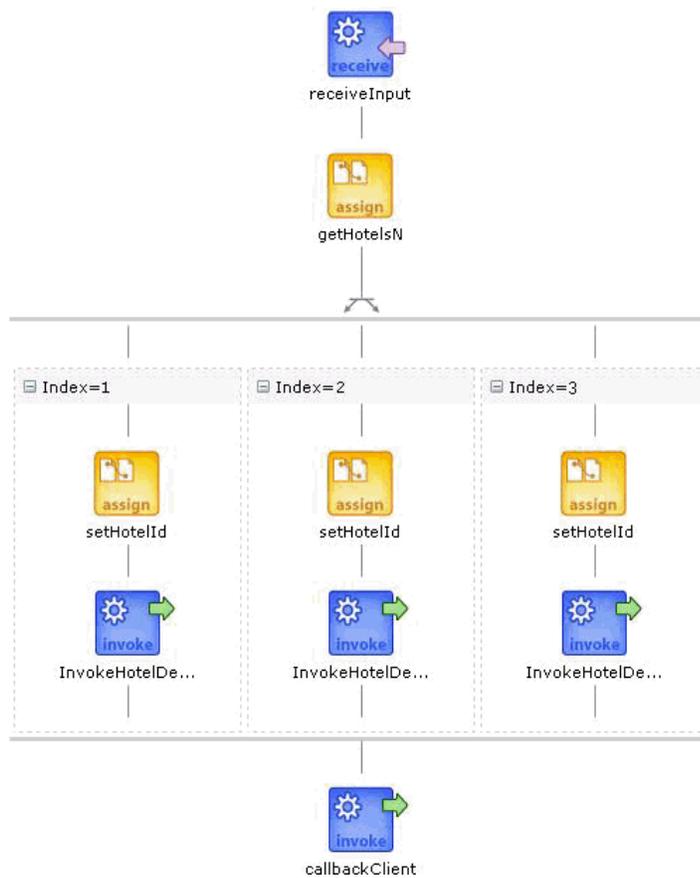
The branches created by flowN perform the same activities, but use different data. Each branch uses the index variable to look up input variables. The index variable can be used in the XPath expression to acquire the data specific for that branch.

For example, suppose there is an array of data. The BPEL process service component uses a count function to determine the number of elements in the array. Then the process sets N to be the number of elements. The index variable starts at a preset value (zero is the default), and flowN creates branches to retrieve each element of the array and perform activities using data contained in that element. These branches are generated and performed in parallel, using all the values between the initial index value and N. flowN terminates when the index variable reaches the value of N. For example, if the array contains 3 elements, N is set to 3. Assuming the index variable begins at 1, the flowN activity creates three parallel branches with indexes 1, 2, and 3.

The flowN activity can use data from other sources as well, including data obtained from web services.

Figure 10–5 shows the runtime flow of a flowN activity in Oracle Enterprise Manager Fusion Middleware Control Console that looks up three hotels. This is different from the view because instead of showing the BPEL process service component, it shows how the process has actually executed. In this case, there are three hotels, but the number of branches changes to match the number of hotels available.

Figure 10–5 Oracle Enterprise Manager Fusion Middleware Control Console View of the Execution of a flowN activity



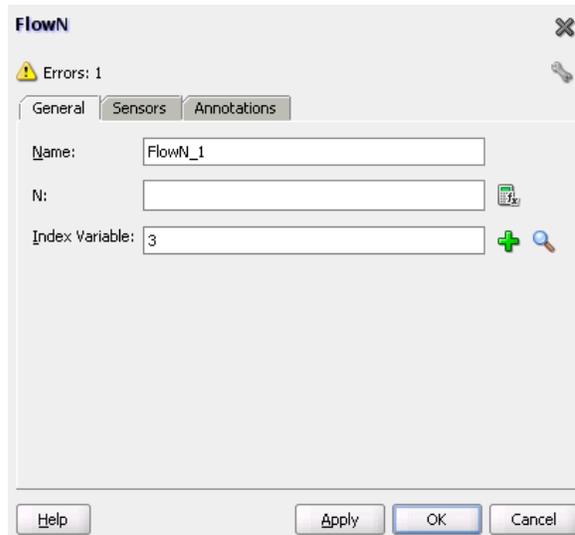
10.3.1 How to Create a flowN Activity

To create a flowN activity:

1. From the Component Palette, drag a **FlowN** activity into the designer.
2. Click the + sign to expand the **FlowN** activity.
3. Double-click the **FlowN** activity.

Figure 10–6 shows the flowN dialog.

Figure 10–6 FlowN Dialog

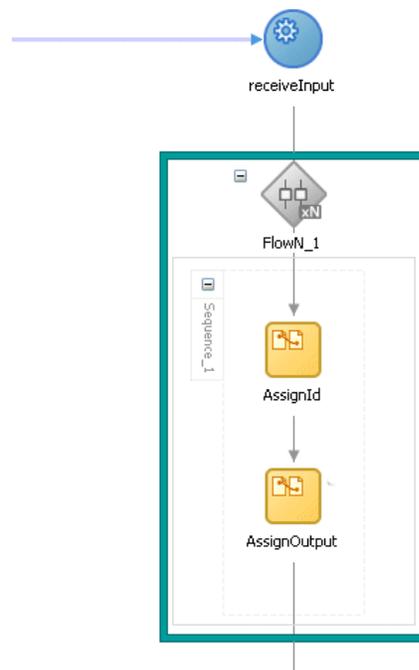


The flowN dialog enables you to:

- Name the activity
 - Enter a value or an expression for calculating the value of N (the number of branches to create)
 - Define the index variable (the time to wait in each branch)
4. Drag and define additional activities in the flowN activity.

Figure 10–7 shows how a FlowN activity appears with additional activities.

Figure 10–7 FlowN Activity with Additional Activities



10.3.2 What Happens When You Create a FlowN Activity

The following code shows the `.bpel` file that uses the `flowN` activity to look up information on an arbitrary number of hotels. The following actions take place.

[Example 10-2](#) shows the sequence name.

Example 10-2 Sequence Name

```
<sequence name="main">
<!-- Received input from requestor.
Note: This maps to operation defined in NflowHotels.wsdl
The requestor send a set of hotels names wrapped into the "inputVariable"
-->
```

A receive activity calls the client partner link to get the information that the `flowN` activity must define `N` times and look up hotel information. [Example 10-3](#) provides an example.

Example 10-3 Receive Activity

```
<receive name="receiveInput" partnerLink="client"
portType="client:NflowHotels" operation="initiate" variable="inputVariable"
createInstance="yes"/>
<!--
The 'count()' Xpath function is used to get the number of hotelName
noded passed in.
An intermediate variable called "NbParallelFlow" is
used to store the number of N flows being executed
-->
<assign name="getHotelsN">
<copy>
<from
expression="count(bpws:getVariableData('inputVariable', 'payload', '/client:Nflow
HotelsProcessRequest/client:ListOfHotels/client:HotelName'))"/>
<to variable="NbParallelFlow"/>
</copy>
</assign>
<!-- Initiating the FlowN activity
The N value is initialized with the value stored in the
"NbParallelFlow" variable
The variable call "Index" is defined as the index variable
NOTE: Both "NbParallelFlow" and "Index" variables have to be declared
-->
```

The `flowN` activity begins next. After defining a name for the activity of `flowN`, `N` is defined as a value from the `inputVariable`, which is the number of hotel entries. The activity also assigns `index` as the index variable. [Example 10-4](#) provides an example.

Example 10-4 FlowN Activity

```
<bpelx:flowN name="FlowN" N="bpws:getVariableData('NbParallelFlow')
indexVariable="Index">
<sequence name="Sequence_1">
<!-- Fetching each hotelName by indexing the "inputVariable" with the
"Index" variable.
Note the usage of the "concat()" Xpath function to create the
expression accessing the array element.
-->
```

The copy rule shown in [Example 10–5](#) then uses the index variable to concatenate the hotel entries into a list:

Example 10–5 Assign Activity

```
<assign name="setHotelId">
  <copy>
    <from expression=
"bpws:getVariableData('inputVariable','payload',concat('/client:NflowHotelsProcessRequest/client:ListOfHotels/client:HotelName['
bpws:getVariableData('Index'),''])")"/>
      <to variable="InvokeHotelDetailInputVariable" part="payload"
query="/ns2:hotelInfoRequest/ns2:id"/>
    </copy>
  </assign>
```

Using the hotel information, an `invoke` activity looks up detailed information for each hotel through a web service. [Example 10–6](#) provides an example.

Example 10–6 Invoke Activity

```
<!-- For each hotel, invoke the web service giving detailed information
on the hotel -->
  <invoke name="InvokeHotelDetail" partnerLink="getHotelDetail"
portType="ns2:getHotelDetail" operation="process"
inputVariable="InvokeHotelDetailInputVariable"
outputVariable="InvokeHotelDetailOutputVariable"/>
  <!-- This process does not do anything with the retrieved information.
In real life, it could then be used to continue the process.
Note: Meanwhile an indexing variable is used. Unlike a while loop, the
activities are executed in parallel, not sequentially.
-->
  </sequence>
</bpelx:flowN>
```

Finally, the BPEL process sends detailed information on each hotel to the client partner link. [Example 10–7](#) provides an example.

Example 10–7 Invoke Activity

```
  <invoke name="callbackClient" partnerLink="client"
portType="client:NflowHotelsCallback" operation="onResult"
inputVariable="outputVariable"/>
  </sequence>
</sequence>
```

Using Conditional Branching in a BPEL Process

This chapter describes how to use conditional branching in a BPEL process service component. Conditional branching introduces decision points to control the flow of execution of a BPEL process service component.

This chapter includes the following sections:

- [Section 11.1, "Introduction to Conditional Branching"](#)
- [Section 11.2, "Creating a Switch Activity to Define Conditional Branching"](#)
- [Section 11.3, "Creating a While Activity to Define Conditional Branching"](#)

For additional information on creating conditional branching in a SOA composite application, see *Oracle Fusion Middleware Tutorial for Running and Building an Application with Oracle SOA Suite*.

11.1 Introduction to Conditional Branching

BPEL applies logic to make choices through conditional branching. You can use either of the following activities to design your code to select different actions based on conditional branching:

- Switch activity
 - Enables you to 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 service component moves to the next branch condition, until it either finds a valid branch condition, encounters an otherwise branch, or runs out of branches. If multiple branch conditions are true, then BPEL executes the first true branch. [Section 11.2, "Creating a Switch Activity to Define Conditional Branching"](#) explains how to create switch activities.
- While activity
 - Enables you to create a while loop to select between two actions. [Section 11.3, "Creating a While Activity to Define Conditional Branching"](#) describes while activities.

Many branches are set up, and each branch has a condition in the form of an XPath expression.

You can program a conditional branch to have a timeout. That is, if a response cannot be generated in a specified period, the BPEL flow can stop waiting and resume its activities. [Chapter 14, "Using Events and Timeouts in BPEL Processes"](#) explains this feature in detail.

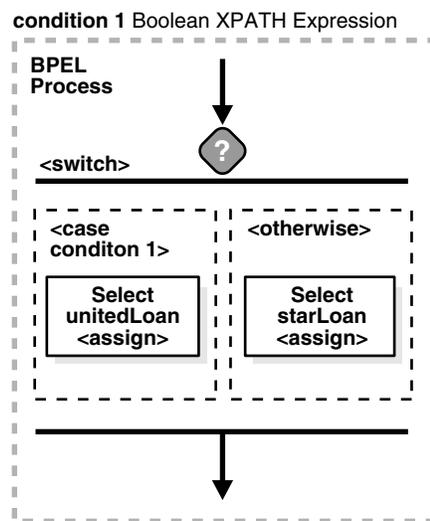
Note: You can also define conditional branching logic with business rules. See *Oracle Fusion Middleware User's Guide for Oracle Business Rules* and *Oracle Fusion Middleware Tutorial for Running and Building an Application with Oracle SOA Suite* for details.

11.2 Creating a Switch Activity to Define Conditional Branching

Assume you designed a flow activity in the BPEL process service component that gathered loan offers from two companies at the same time, but did not compare either of the offers. Each offer was stored in its own global variable. To compare the two bids and make decisions based on that comparison, you can use a switch activity.

Figure 11-1 provides an overview of a BPEL conditional branching process that has been defined in a switch activity.

Figure 11-1 Conditional Branching

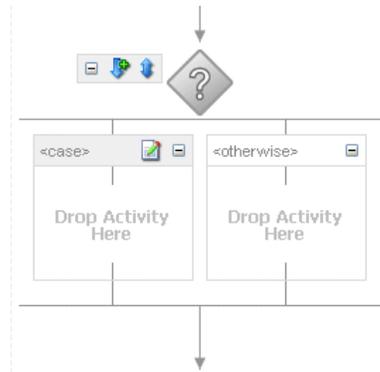


11.2.1 How to Create a Switch Activity

To create a switch activity:

1. From the Component Palette, drag a **Switch** activity into the designer.
2. Click the + sign to expand the **switch** activity, as shown in Figure 11-2.

The **Switch** activity has two switch case branches by default, each with a box for functional elements. If you want to add more branches, select the entire switch activity, right-click, and select **Add Switch Case** from the menu.

Figure 11–2 Switch Activity

3. In the first branch, right-click and select **Edit** from the menu.
The Switch Case dialog appears.
4. In the **Expression** field, enter an XPath boolean expression by pressing Ctrl+Space to start the XPath Building Assistant. [Example 11–1](#) provides details.

Example 11–1 XPath Expression

```
bpws:getVariableData('loanOffer1','payload','/loanOffer/APR') >
bpws:getVariableData('loanOffer2','payload','/loanOffer/APR')
```

5. Enter this expression on one line. To use the XPath Expression Builder, click the **XPath Expression Builder** icon above the **Expression** field.

In this example, two loan offers from completing loan companies are stored in the global variables `loanOffer1` and `loanOffer2`. Each loan offer variable contains the loan offer's APR. The BPEL flow must choose the loan with the lower APR.

One of the following switch activities takes place:

- If `loanOffer1` has the higher APR, then the first branch selects `loanOffer2` by assigning the `loanOffer2` payload to the `selectedLoanOffer` payload.
- If `loanOffer1` does *not* have the lower APR than `loanOffer2`, the otherwise case assigns the `loanOffer1` payload to the `selectedLoanOffer` payload.

11.2.2 What Happens When You Create a Switch Activity

A switch activity, like a flow activity, has multiple branches. In [Example 11–2](#), there are only two branches shown in the `.bpel` file after design completion. The first branch, which selects a loan offer from a company named United Loan, is executed if a case condition containing an XPath boolean expression is met. Otherwise, the second branch, which selects the offer from a company named Star Loan, is executed. By default, the switch activity provides two switch cases, but you can add more if you want.

Example 11–2 Switch Activity

```
<switch name="switch-1">
  <case condition="bpws:getVariableData('loanOffer1','payload',
    '/autoloan:loanOffer/autoloan:APR') <
    bpws:getVariableData('loanOffer2','payload','/autoloan:loanOffer/autoloan:APR
    ')">
    <assign name="selectUnitedLoan">
```

```

        <copy>
          <from variable="loanOffer1" part="payload">
            </from>
          <to variable="selectedLoanOffer" part="payload"/>
        </copy>
      </assign>
    </case>
    <otherwise>
      <assign name="selectStarLoan">
        <copy>
          <from variable="loanOffer2" part="payload">
            </from>
          <to variable="selectedLoanOffer" part="payload"/>
        </copy>
      </assign>
    </otherwise>
  </switch>

```

11.3 Creating a While Activity to Define Conditional Branching

Another way to design your BPEL code to select between multiple actions is to use a while activity to create a while loop. The while loop repeats an activity until a specified success criteria is met. For example, if a critical web service is returning a service busy message in response to requests, you can use the while activity to keep polling the service until it becomes available. The condition for the while activity is that the latest message received from the service is busy, and the operation within the while activity is to check the service again. Once the web service returns a message other than service busy, the while activity terminates and the BPEL process service component continues, ideally with a valid response from the web service.

11.3.1 How To Create a While Activity

To create a while activity:

1. From the Component Palette, drag a **While** activity into the designer.
2. Click the + sign to expand the while activity.

The while activity has icons to allow you to build condition expressions and to validate the while definition. It also provides an area for you to drag an activity to define the while loop. [Figure 11–3](#) provides an example.

Figure 11–3 While Activity

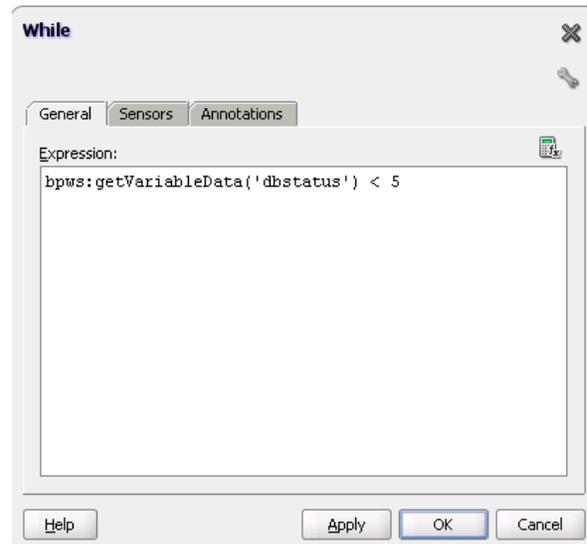


3. Drag and define additional activities for using the while condition into the **Drop Activity Here** area of the **While** activity (for example, a **Scope** activity).

The activities can be existing or new activities.

4. Press Ctrl+Space to invoke the XPath Building Assistant or click the **XPath Expression Builder** icon to open the Expression Builder dialog.
5. Enter an expression to perform repeatedly, as shown in [Figure 11-4](#). This action is performed until the given boolean while condition is no longer true. In this example, this activity is set to loop while less than 5.

Figure 11-4 While Activity with an Expression



6. Click **OK** when complete.

11.3.2 What Happens When You Create a While Activity

[Example 11-3](#) provides an example of the `.bpel` file after design completion. The while activity includes a scope activity. The scope activity includes invoke, assign, and wait activities. Database exception handling tasks are performed by creating a local variable and placing the invoke activity inside the scope activity. The local variable is set to false (represented by 0). You attempt to call the external partner service in the while loop activity until the local variable is satisfied (set to 1). The while activity is set to loop a maximum of five times. In the case of an exception, you reset the flag to false (0).

Example 11-3 While Activity

```
<while name="While_1" condition="bpws:getVariableData('dbStatus') > 5">
  <scope name="Scope_1">
<faultHandlers>
  <catchAll>
    <sequence name="Sequence_2">
      <assign name="assign_DB_retry">
        <copy>
          <from expression="bpws:getVariableData('dbStatus') + 1"/>
          <to variable="dbStatus"/>
        </copy>
      </assign>
      <wait name="Wait_30_sec" for="'PT31S'"/>
    </sequence>
  </catchAll>
</faultHandlers>
```

```
<sequence name="Sequence_1">
  <invoke name="Write_DBWrite" partnerLink="WriteDBRecord"
    portType="ns2:WriteDBRecord_ptt" operation="insert"
    inputVariable="Invoke_DBWrite_merge_InputVariable"/>
  <assign name="Assign_dbComplete">
    <copy>
      <from expression="'10'"/>
      <to variable="dbStatus"/>
    </copy>
  </assign>
</sequence>
</scope>
</while>
```

Using Fault Handling in a BPEL Process

This chapter describes how to use fault handling in a BPEL process. Fault handling allows a BPEL process service component to handle error messages or other exceptions returned by outside web services, and to generate error messages in response to business or runtime faults. You can also define a fault management framework to catch faults and perform user-specified actions defined in a fault policy file.

This chapter includes the following sections:

- [Section 12.1, "Introduction to a Fault Handler"](#)
- [Section 12.2, "Introduction to BPEL Standard Faults"](#)
- [Section 12.3, "Introduction to Categories of BPEL Faults"](#)
- [Section 12.4, "Using the Fault Management Framework"](#)
- [Section 12.5, "Catching BPEL Runtime Faults"](#)
- [Section 12.6, "Getting Fault Details with the getFaultAsString XPath Extension Function"](#)
- [Section 12.7, "Throwing Internal Faults"](#)
- [Section 12.8, "Returning External Faults"](#)
- [Section 12.9, "Using a Scope Activity to Manage a Group of Activities"](#)
- [Section 12.10, "Using Compensation After Undoing a Series of Operations"](#)
- [Section 12.11, "Using the Terminate Activity to Stop a Business Process Instance"](#)

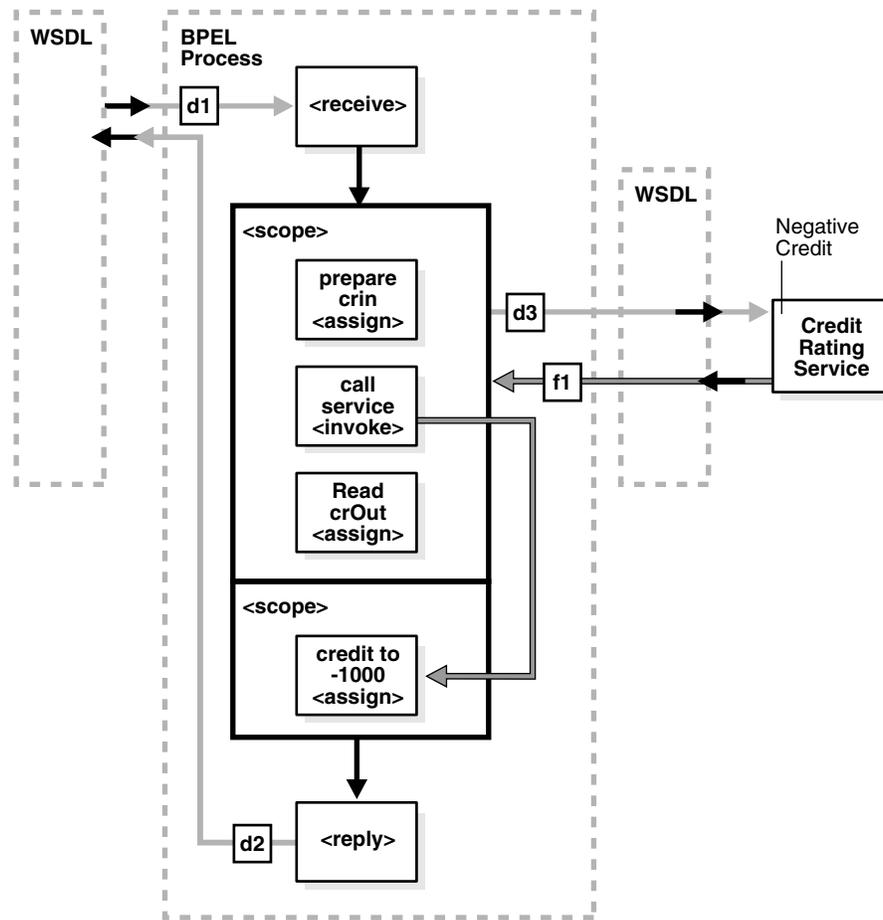
For additional information on creating fault handling in a SOA composite application, see *Oracle Fusion Middleware Tutorial for Running and Building an Application with Oracle SOA Suite*.

12.1 Introduction to a Fault Handler

Fault handlers define how the BPEL process service component responds when the web services return data other than what is normally expected (for example, returning an error message instead of a number). An example of a fault handler is where the web service normally returns a credit rating number, but instead returns a negative credit message.

[Figure 12-1](#) provides an example of how a fault handler sets a credit rating variable to -1000.

Figure 12-1 Fault Handling



The code segment in [Example 12-1](#) defines the fault handler for this operation in the BPEL file:

Example 12-1 Fault Handler Definition

```
<faultHandlers>
  <catch faultName="services:NegativeCredit" faultVariable="crError">
    <assign name="crin">
      <copy>
        <from expression="-1000">
        </from>
        <to variable="input" part="payload"
          query="/autoloan:loanApplication/autoloan:creditRating"/>
      </copy>
    </assign>
  </catch>
</faultHandlers>
```

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`.

When you select web services for the BPEL process service component, determine the possible faults that may be returned and set up a fault handler for each one.

12.2 Introduction to BPEL Standard Faults

The *Business Process Execution Language for Web Services Specification* defines the following standard faults in the namespace of `http://schemas.xmlsoap.org/ws/2003/03/business-process/`:

- `bindingFault`
- `conflictingReceive`
- `conflictingRequest`
- `correlationViolation`
- `forcedTermination`
- `invalidReply`
- `joinFailure`
- `mismatchedAssignmentFailure`
- `remoteFault`
- `repeatedCompensation`
- `selectionFailure`
- `uninitializedVariable`

Standard faults are defined as follows:

- Typeless, meaning they do not have associated `messageTypes`
- Not associated with any Web Services Description Language (WSDL) message
- Caught without a fault variable:

```
<catch faultName="bpws:selectionFailure">
```

12.3 Introduction to Categories of BPEL Faults

A BPEL fault has a fault name called a *Qname* (name qualified with a namespace) and a possible `messageType`. There are two categories of BPEL faults:

- Business faults
- Runtime faults

12.3.1 Business Faults

Business faults are application-specific faults that are generated when there is a problem with the information being processed (for example, when a social security number is not found in the database). A business fault occurs when an application executes a `throw` activity or when an `invoke` activity receives a fault as a response. The fault name of a business fault is specified by the BPEL process service component. The `messageType`, if applicable, is defined in the WSDL. A business fault can be caught with a `faultHandler` using the `faultName` and a `faultVariable`.

```
<catch faultName="ns1:faultName" faultVariable="varName">
```

12.3.2 Runtime Faults

Runtime faults are the result of problems within the running of the BPEL process service component or web service (for example, data cannot be copied properly

because the variable name is incorrect). These faults are not user-defined, and are thrown by the system. They are generated if the process tries to use a value incorrectly, a logic error occurs (such as an endless loop), a Simple Object Access Protocol (SOAP) fault occurs in a SOAP call, an exception is thrown by the server, and so on.

Several runtime faults are automatically provided. These faults are included in the `http://schemas.oracle.com/bpel/extension` namespace. These faults are associated with the `messageType RuntimeFaultMessage`. The WSDL file shown in [Example 12-2](#) defines the `messageType`:

Example 12-2 *messageType Definition*

```
<?xml version="1.0" encoding="UTF-8" ?>
<definitions name="RuntimeFault"
  targetNamespace="http://schemas.oracle.com/bpel/extension"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns="http://schemas.xmlsoap.org/wsdl/">

  <message name="RuntimeFaultMessage">
    <part name="code" type="xsd:string" />
    <part name="summary" type="xsd:string" />
    <part name="detail" type="xsd:string" />
  </message>
</definitions>
```

If a `faultVariable` (of `messageType RuntimeFaultMessage`) is used when catching the fault, the fault code can be queried from the `faultVariable`, along with the fault summary and detail.

12.3.2.1 **bindingFault**

A `bindingFault` is thrown inside an activity if the preparation of the invocation fails. For example, the WSDL of the process fails to load. A `bindingFault` is not retryable. This type of fault usually must be fixed by human intervention.

12.3.2.2 **remoteFault**

A `remoteFault` is also thrown inside an activity. It is thrown because the invocation fails. For example, a SOAP fault is returned by the remote service.

12.3.2.3 **replayFault**

A `replayFault` replays the activity inside a scope. At any point inside a scope, this fault is migrated up to the scope. The server then re-executes the scope from the beginning.

12.4 Using the Fault Management Framework

Oracle SOA Suite provides a generic fault management framework for handling faults in BPEL processes. If a fault occurs during runtime in an invoke activity in a process, the framework catches the fault and performs a user-specified action defined in a fault policy file associated with the activity. If a fault results in a condition in which human intervention is the prescribed action, you perform recovery actions from Oracle Enterprise Manager Fusion Middleware Control Console. The fault management framework provides an alternative to designing a BPEL process with catch activities in scope activities.

This section provides an overview of the components that comprise the fault management framework.

- The fault management framework catches all faults (business and runtime) for an invoke activity.
- A fault policy file defines fault conditions and their corresponding fault recovery actions. Each fault condition specifies a particular fault or group of faults, which it attempts to handle, and the corresponding action for it. A set of actions is identified by an ID in the fault policy file.
- A set of conditions invokes an action (known as fault policy).
- A fault policy bindings file associates the policies defined in the fault policy file with the following:
 - SOA composite applications
 - BPEL process and Oracle Mediator service components
 - Reference binding components for BPEL process and Oracle Mediator service components

The framework looks for fault policy bindings in the same directory as the `composite.xml` file of the SOA composite application or in a remote location identified by two properties that you set.

Note: A fault policy configured with the fault management framework overrides any fault handling defined in catch activities of scope activities in the BPEL process. The fault management framework can be configured to rethrow the fault handling back to the catch activities.

- The fault policy file (`fault-policies.xml`) and fault policy bindings file (`fault-bindings.xml`) are placed in either of the following locations:
 - In the same directory as the `composite.xml` file of the SOA composite application.
 - In a different location that is specified with two properties that you add to the `composite.xml` file. This option is useful if a fault policy must be used by multiple SOA composite applications. This option overrides any fault policy files that are included in the same directory as the `composite.xml` file. [Example 12-3](#) provides details about these two properties. In this example, the fault policy files are placed into the SOA Metadata Service (MDS) shared area.

Example 12-3 *Fault Policies used by Multiple SOA Composite Applications*

```
<property
  name="oracle.composite.faultPolicyFile">oramds://apps/faultpolicyfiles/
  fault-policies.xml
</property>
<property
  name="oracle.composite.faultBindingFile">oramds://apps/faultpolicyfiles/
  fault-bindings.xml
</property>
```

See [Chapter 20, "Using Mediator Error Handling"](#) for details about Oracle Mediator fault handling capabilities.

12.4.1 How to Design a Fault Policy

This section describes how to design a fault policy.

Note: The Facades API enables you to programmatically perform the abort, retry (with a success action), continue, rethrow, and replay recovery options.

12.4.1.1 Understanding How Fault Policy Binding Resolution Works

A fault policy bindings file associates the policies defined in a fault policy file with the SOA composite application or the component (service component or reference binding component). The framework attempts to identify a fault policy binding in the following order:

- Reference binding component defined in the `composite.xml` file.
- BPEL process or Oracle Mediator service component defined in the `composite.xml` file.
- SOA composite application defined in the `composite.xml` file.

During the resolution process, if no action is found that matches the condition, the framework assumes that resolution failed and moves to the next resolution level.

For example, assume an invoke activity faults with `faultname="abc"`. There is a policy binding specified in the `fault-binding.xml` file:

- SOA composite application binds to `policy-id-1`
- BPEL process or Oracle Mediator service component or reference binding component binds to `policy-id-2`

In the `fault-bindings.xml` file, the following bindings are also specified:

- SOA composite application binds to `policy-id-3`
- Reference binding component or service component binds to `policy-id-4`

The fault management framework behaves as follows:

- First match the resolve binding (in this case, `policy-id-2`).
- If the fault resolution fails, go to the next possible match (`policy-id-4`).
- If the fault resolution fails, go to the next possible match (`policy-id-3`).
- If the fault resolution fails, go to the next possible match (in this case, `policy-id-1`).
- If the fault resolution still fails, the fault is sent to the BPEL fault catch activity.

12.4.1.2 Creating a Fault Policy File for Automated Fault Recovery

1. Create a fault policy file (for example, named `fault-policies.xml`). This file includes `condition` and `action` sections for performing specific tasks.
2. Place the file in the same directory as the `composite.xml` file or place it in a different location and define the `oracle.composite.faultPolicyFile` property. [Example 12-4](#) provides details.

Example 12-4 Defining Properties

```
<property
  name="oracle.composite.faultPolicyFile">orams://apps/faultpolicyfiles/
  fault-policies.xml
</property>
<property
```

```

name="oracle.composite.faultBindingFile">oramds://apps/faultpolicyfiles/
fault-bindings.xml
</property>

```

3. Define the condition section of the fault policy file.

- Note the following details about the condition section:
 - This section provides a condition based on `faultName`.
 - Multiple conditions may be configured for a `faultName`.
 - Each condition has one `test` section (an XPath expression) and one `action` section.
 - The `test` section (XPath expression) is evaluated for the fault variable available in the fault.
 - The `action` section has a reference to the action defined in the same file.
 - You can only query the fault variable available in the fault.
 - The order of condition evaluation is determined by the sequential order in the document.

Table 12–1 provides examples of condition section use in the fault policy file. All actions defined in the condition section must be associated with an action in the action section.

Table 12–1 Use of the condition Section in the Fault Policy File

Condition Example	Fault Policy File Syntax
This condition is checking a fault variable for code = "WSDLFailure"	<pre> <condition> <test>\$fault.code="WSDLReading Error" </test> </pre>
An action of <code>ora-terminate</code> is specified.	<pre> <action ref="ora-terminate"/> </condition> </pre>
No test condition is provided. This is a catch all condition for a given <code>faultName</code> .	<pre> <condition> <action ref="ora-rethrow"/> </condition> </pre>
If the <code>faultName</code> name attribute is missing, this indicates a catch all activity for faults that have any <code>QName</code> .	<pre> <faultName > . . . </faultName> </pre>

- ### 4. Define the action section of the fault policy file. Note that validation of fault policy files is done during deployment. If you change the fault policy, you must redeploy the SOA composite application that includes the fault policy.

Table 12–2 provides several examples of action section use in the fault policy file. You can provide automated recovery actions for some faults. In all recovery actions except retry and human intervention, the framework performs the actions synchronously.

Table 12–2 Use of action Section in the Fault Policy File

Recovery Actions	Fault Policy File Syntax
<p>Retry: Provides the following actions for retrying the activity.</p> <ul style="list-style-type: none"> ■ Retry a specified number of times. ■ Provide a delay between retries (in seconds). ■ Increase the interval with an exponential back off. ■ Chain to a retry failure action if retry <i>N</i> times fails. ■ Chain to a retry success action if a retry is successful. <p>Note: Exponential back off indicates the next retry attempt is scheduled at 2 x the <i>delay</i>, where <i>delay</i> is the current retry interval. For example, if the current retry interval is 2 seconds, the next retry attempt is scheduled at 4, the next at 8, and the next at 16 seconds until the <code>retryCount</code> value is reached.</p>	<pre><Action id="ora-retry"> <Retry> <retryCount>3</retryCount> <retryInterval>2</retryInterval> <exponentialBackoff/> <retryFailureAction ref="ora-java"/> <retrySuccessAction ref="ora-java"/> </Retry> </Action></pre> <p>Note the following details:</p> <ul style="list-style-type: none"> ■ The framework chains to the retry success action if the retry attempt is successful. ■ If all retry attempts fail, the framework chains to the retry failure action.
<p>Human Intervention: Causes the current activity to stop processing. You can now go to Oracle Enterprise Manager Fusion Middleware Control Console and perform manual recovery actions on this instance.</p>	<pre><Action id="ora-human-intervention"> <humanIntervention/></Action></pre>
<p>Terminate Process: Terminates the process</p>	<pre><Action id="ora-terminate"><abort/></Action></pre>
<p>Java Code: Enables you to execute an external Java class.</p> <p><code>returnValue</code>: The implemented Java class must implement a method that returns a string. The policy can chain to a new action based on the returned string.</p> <p>For additional information, see Section 12.4.3, "How to Use a Java Action Fault Policy"</p>	<pre><Action id="ora-java"> <!-- this is user provided custom java class--> <javascriptAction className="mypackage.myClass" defaultAction="ora-terminate"> <returnValue value="REPLAY" ref="ora-terminate"/> <returnValue value="RETHROW" ref="ora-rethrow-fault"/> <returnValue value="ABORT" ref="ora-terminate"/> <returnValue value="RETRY" ref="ora-retry"/> <returnValue value="MANUAL" ref="ora-human-intervention"/> </javascriptAction> </Action></pre>
<p>Rethrow Fault: The framework sends the fault to the BPEL fault handlers (catch activities in scope activities). If none are available, the fault is sent up.</p>	<pre><Action id="ora-rethrow-fault"><rethrowFault/></Action></pre>
<p>Replay Scope: Raises a replay fault.</p>	<pre><Action id="ora-replay-scope"><replayScope/></Action></pre>

Note: The preseeded recovery action tag names (`ora-retry`, `ora-human-intervention`, `ora-terminate`, and so on) are only samples. You can substitute these names with ones appropriate to your environment.

[Example 12-5](#) shows a fault policy file with fully-defined condition and action sections.

Notes:

- Fault policy file names are not restricted to one specific name. However, they must conform to the `fault-policy.xsd` schema file.
- [Example 12-5](#) provides an example of catching faults based on fault names. You can also catch faults based on message types, or on both:

```
<fault name="myfault" type="fault:faultType">
```

Example 12-5 Fault Policy File

```
<?xml version="1.0" encoding="UTF-8"?>
<faultPolicies xmlns="http://schemas.oracle.com/bpel/faultpolicy"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <faultPolicy version="0.0.1" id="FusionMidFaults"
xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns="http://schemas.oracle.com/bpel/faultpolicy"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <Conditions>
      <faultName xmlns:medns="http://schemas.oracle.com/mediator/faults"
name="medns:mediatorFault">
        <condition>
          <action ref="MediatorJavaAction"/>
        </condition>
      </faultName>
      <faultName xmlns:bpelx="http://schemas.oracle.com/bpel/extension"
name="bpelx:remoteFault">
        <condition>
          <action ref="BPELJavaAction"/>
        </condition>
      </faultName>
      <faultName xmlns:bpelx="http://schemas.oracle.com/bpel/extension"
name="bpelx:bindingFault">
        <condition>
          <action ref="BPELJavaAction"/>
        </condition>
      </faultName>
      <faultName xmlns:bpelx="http://schemas.oracle.com/bpel/extension"
name="bpelx:runtimeFault">
        <condition>
          <action ref="BPELJavaAction"/>
        </condition>
      </faultName>
    </Conditions>
    <Actions>
      <!-- Generics -->
```

```

    <Action id="default-terminate">
      <abort/>
    </Action>
    <Action id="default-replay-scope">
      <replayScope/>
    </Action>
    <Action id="default-rethrow-fault">
      <rethrowFault/>
    </Action>
    <Action id="default-human-intervention">
      <humanIntervention/>
    </Action>
    <Action id="MediatorJavaAction">
      <!-- this is user provided class-->
      <javaAction className="MediatorJavaAction.myClass"
defaultAction="default-terminate">
        <returnValue value="MANUAL" ref="default-human-intervention"/>
      </javaAction>
    </Action>
    <Action id="BPELJavaAction">
      <!-- this is user provided class-->
      <javaAction className="BPELJavaAction.myAnotherClass"
defaultAction="default-terminate">
        <returnValue value="MANUAL" ref="default-human-intervention"/>
      </javaAction>
    </Action>
  </Actions>
</faultPolicy>
</faultPolicies>

```

12.4.1.3 Associating a Fault Policy with Fault Policy Binding

Note: The fault policy file binding file must be named `fault-bindings.xml`. This conforms to the `fault-bindings.xsd` schema file.

1. Create a fault policy binding file (`fault-bindings.xml`) that associates the policies defined in the fault policy file with the level of fault policy binding you are using (either a SOA composite application or a component (reference binding component or BPEL process or Oracle Mediator service component)).
2. Place the file in the same directory as the `composite.xml` file or place it in a remote location and define the `oracle.composite.faultBindingFile` property as shown in Step 2 on page 12-6.

[Example 12-6](#) shows a fault policy bindings file that associates the fault policies defined in the `fault-policies.xml` file with the `FusionMidFaults` SOA composite application.

Example 12-6 *fault-buildings.xml* File

```

<?xml version="1.0" encoding="UTF-8" ?>
<faultPolicyBindings version="0.0.1"
  xmlns="http://schemas.oracle.com/bpel/faultpolicy"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <composite faultPolicy="FusionMidFaults"/>
  <!--<composite faultPolicy="ServiceExceptionFaults"/>-->
  <!--<composite faultPolicy="GenericSystemFaults"/>-->

```

```
</faultPolicyBindings>
```

12.4.1.4 Additional Fault Policy and Fault Policy Binding File Samples

This section provides additional samples of fault policy and fault policy binding files.

[Example 12-7](#) shows the `fault-policies.xml` file contents.

Example 12-7 *fault-policies.xml* File

```
<?xml version="1.0" encoding="UTF-8"?>
<faultPolicies xmlns="http://schemas.oracle.com/bpel/faultpolicy">
<faultPolicy version="2.0.1"
    id="CRM_ServiceFaults"
    xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns="http://schemas.oracle.com/bpel/faultpolicy"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <Conditions>
    <!-- Fault if wsdlRuntimeLocation is not reachable -->
    <faultName xmlns:bpelx="http://schemas.oracle.com/bpel/extension"
name="bpelx:remoteFault">
    <condition>
    <test>$fault.code="WSDLReadingError"</test>
    <action ref="ora-terminate"/>
    </condition>
    <condition>
    <action ref="ora-java"/>
    </condition>
    </faultName>
    <!-- Fault if location port is not reachable-->
    <faultName xmlns:bpelx="http://schemas.oracle.com/bpel/extension"
name="bpelx:bindingFault">
    <!--ORA-00001: unique constraint violated on insert-->
    <condition>
    <test>$fault.code="1"</test>
    <action ref="ora-java"/>
    </condition>
    <!--ORA-01400: cannot insert NULL -->
    <condition>
    <test xmlns:test="http://test">$fault.code="1400"</test>
    <action ref="ora-terminate"/>
    </condition>
    <!--ORA-03220: required parameter is NULL or missing -->
    <condition>
    <test>$fault.code="3220"</test>
    <action ref="ora-terminate"/>
    </condition>
    <condition>
    <action ref="ora-retry-crm-endpoint"/>
    </condition>
    </faultName>
    <!-- Business faults -->
    <!-- Fault comes with a payload of error, make sure the name space is
provided here or at root level -->
    <faultName xmlns:credit="http://services.otn.com"
name="credit:NegativeCredit">
    <!-- we get this fault when SSN starts with 0-->
    <condition>
    <test>$fault.payload="Bankruptcy Report"</test>
```

```

        <action ref="ora-human-intervention"/>
        <!--action ref="ora-retry"/-->
    </condition>
    <!-- we get this fault when SSN starts with 1-->
    <condition>
        <test>$fault.payload="Bankruptcy Report-abort"</test>
        <action ref="ora-terminate"/>
    </condition>
    <!-- we get this fault when SSN starts with 2-->
    <condition>
        <test>$fault.payload="Bankruptcy Report-rethrow"</test>
        <action ref="ora-rethrow-fault"/>
    </condition>
    <!-- we get this fault when SSN starts with 3-->
    <condition>
        <test>$fault.payload="Bankruptcy Report-replay"</test>
        <action ref="ora-replay-scope"/>
    </condition>
    <!-- we get this fault when SSN starts with 4-->
    <condition>
        <test
xmlns:myError="http://services.otn.com">$fault.payload="Bankruptcy
Report-human"</test>
        <action ref="ora-human-intervention"/>
    </condition>
    <!-- we get this fault when SSN starts with 5-->
    <condition>
        <test>$fault.payload="Bankruptcy Report-java"</test>
        <action ref="ora-java"/>
    </condition>
</faultName>

    </Conditions>
    <Actions>
        <Action id="ora-retry">
    <retry>
        <retryCount>3</retryCount>
        <retryInterval>2</retryInterval>
        <exponentialBackoff/>
        <retryFailureAction ref="ora-java"/>
        <retrySuccessAction ref="ora-java"/>
    </retry>
    </Action>
    <Action id="ora-retry-crm-endpoint">
    <retry>
        <retryCount>5</retryCount>
        <retryFailureAction ref="ora-java"/>
        <retryInterval>5</retryInterval>
        <retrySuccessAction ref="ora-java"/>
    </retry>
    </Action>
    <Action id="ora-replay-scope">
    <replayScope/>
    </Action>
    <Action id="ora-rethrow-fault">
    <rethrowFault/>
    </Action>
    <Action id="ora-human-intervention">
    <humanIntervention/>
    </Action>

```

```

        <Action id="ora-terminate">
            <abort/>
        </Action>
        <Action id="ora-java">
            <!-- this is user provided class-->
            <javaAction
                className="com.oracle.bpel.client.config.faultpolicy.TestJavaAction"
                defaultAction="ora-terminate" propertySet="prop-for-billing">
                <returnValue value="REPLAY" ref="ora-terminate"/>
                <returnValue value="RETRHOW" ref="ora-rethrow-fault"/>
                <returnValue value="ABORT" ref="ora-terminate"/>
                <returnValue value="RETRY" ref="ora-retry"/>
                <returnValue value="MANUAL" ref="ora-human-intervention"/>
            </javaAction>
        </Action>

        </Actions>
        <Properties>
            <propertySet name="prop-for-billing">
                <property name="user_email_recipient">bpeladmin</property>
                <property name="email_recipient">joe@abc.com</property>
                <property name="email_recipient">mike@xyz.com</property>
                <property name="email_threshold">10</property>
                <property name="sms_recipient">+429876547</property>
                <property name="sms_recipient">+4212345</property>
                <property name="sms_threshold">20</property>
                <property name="user_email_recipient">john</property>
            </propertySet>
            <propertySet name="prop-for-order">
                <property name="email_recipient">john@abc.com</property>
                <property name="email_recipient">jill@xyz.com</property>
                <property name="email_threshold">10</property>
                <property name="sms_recipient">+42222</property>
                <property name="sms_recipient">+423335</property>
                <property name="sms_threshold">20</property>
            </propertySet>
        </Properties>
    </faultPolicy>
    <faultPolicy version="2.0.1"
        id="Billing_ServiceFaults"
        xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
        xmlns:xs="http://www.w3.org/2001/XMLSchema"

        xmlns="http://schemas.oracle.com/bpel/faultpolicy"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
        <Conditions>
            <faultName>
                <condition>
                    <action ref="ora-manual"/>
                </condition>
            </faultName>
        </Conditions>
        <Actions>
            <Action id="ora-manual">
                <humanIntervention/>
            </Action>
        </Actions>
    </faultPolicy>
</faultPolicies>

```

[Example 12-8](#) shows the `fault-buildings.xml` file that associates the fault policies defined in `fault-policies.xml`.

Example 12-8 Fault Policy Bindings File

```
<?xml version="1.0" encoding="UTF-8"?>
<faultPolicyBindings version="2.0.1"
  xmlns="http://schemas.oracle.com/bpel/faultpolicy"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <composite faultPolicy="ConnectionFaults">
    <component faultPolicy="ServiceFaults">
      <name>Component1</name>
      <name>Component2</name>
    </component>
    <!-- Below listed component names use polic CRM_SeriveFaults -->
    <component faultPolicy="CRM_ServiceFaults">
      <name>HelloWorld</name>
      <name>ShippingComponent</name>
      <name>AnotherComponent "</name>
    </component>
    <!-- Below listed reference names and port types use polic CRM_ServiceFaults
    -->
    <reference faultPolicy="CRM_ServiceFaults">
      <name>creditRatingService</name>
      <name>anotherReference</name>
      <portType
xmlns:credit="http://services.otn.com">credit:CreditRatingService</portType>
      <portType
xmlns:db="http://xmlns.oracle.com/pcbpel/adapter/db/insert/">db:insert_
plt</portType>
    </reference>
    <reference faultPolicy="test1">
      <name>CreditRating3</name>
    </reference>
  </faultPolicyBindings>
```

12.4.1.5 Designing a Fault Policy with Multiple Rejection Handlers

If you design a fault policy that uses the action handler for rejected messages, note that only one write action can be performed. Multiple write actions cannot be performed, even if you define multiple rejection handlers, as shown in [Example 12-9](#). In this case, only the first rejection handler defined (for this example, `ora-queue`) is executed.

Example 12-9 Fault Policy with Multiple Rejection Handlers

```
<faultName xmlns:rjm="http://schemas.oracle.com/sca/rejectedmessages"
name="rjm:FileIn">
  <condition>
    <action ref="ora-queue"/>
  </condition>
</faultName>
<faultName xmlns:rjm="http://schemas.oracle.com/sca/rejectedmessages"
name="rjm:FileIn">
  <condition>
    <action ref="ora-file"/>
  </condition>
</faultName>
```

12.4.2 How to Execute a Fault Policy

You deploy a fault policy as part of a SOA composite application. After deployment, you can perform the following fault recovery actions from Oracle Enterprise Manager Fusion Middleware Control Console:

- Retry the activity
- Modify a variable (available to the faulted activity)
- Continue the instance (mark the activity as a success)
- Rethrow the exception
- Abort the instance
- Throw a replay scope exception

For additional information, see *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite* for the following:

- Instructions on executing a fault policy in Oracle Enterprise Manager Fusion Middleware Control Console
- Use cases in which you define a fault policy that uses human intervention

12.4.3 How to Use a Java Action Fault Policy

Note the following details when using the Java action fault policy:

- The Java class provided follows a specific interface. This interface returns a string. Multiple values can be provided for output and fault policy to take after execution.
- Additional fault policy can be executed by providing a mapping from the output value (return value) of implemented methods to a fault policy.
- If no `ReturnValue` is specified, the default fault policy is executed, as shown in [Example 12–10](#).

Example 12–10 Java Action Fault Policy

```
<Action id="ora-java">
  <JavaAction ClassName="mypackage.myclass"
    defaultAction="ora-human-intervention" propertySet="prop-for-billing">
    <!--defaultAction is a required attribute, but propertySet is optional-->
    <!-- attribute-->
    <ReturnValue value="RETRY" ref="ora-retry"/>
    <!--value is not nilable attribute & cannot be empty-->
    <ReturnValue value="RETRHOW" ref="ora-rethrow-fault"/>
  </JavaAction>
</Action>
```

[Table 12–3](#) provides an example of `ReturnValue` use.

Table 12–3 System Interpretation of Java Action Fault Policy

Code	Description
<code><ReturnValue value="RETRY" ref="ora-retry"/></code>	Execute the <code>ora-retry</code> action if the method returns a string of <code>RETRY</code> .
<code><ReturnValue value="" ref="ora-rethrow"/></code>	Fails in validation.

Table 12–3 (Cont.) System Interpretation of Java Action Fault Policy

Code	Description
<pre><JavaAction ClassName="mypackage.myclass" defaultAction="ora-human-intervention "></pre>	Execute ora-human-intervention after Java code execution. This attribute is used if the return from the method does not match any provided ReturnValue.
<pre><ReturnValue value="RETRY" ref="ora-retry"/> <ReturnValue value="" ref="" /></pre>	Fails in validation.
<pre><JavaAction ClassName="mypackage.myclass" defaultAction=" ora-human-intervention"> <ReturnValue></ReturnValue></pre>	Fails in validation.

To invoke a Java class, you can provide a class that implements the `IFaultRecoveryJavaClass` interface. This interface has two methods, as shown in [Example 12–11](#).

Example 12–11 Implementation of `IFaultRecoveryJavaClass`

```
public interface IFaultRecoveryJavaClass
{
public void handleRetrySuccess( IFaultRecoveryContext ctx );
public String handleFault( IFaultRecoveryContext ctx );
}
```

Note the following details:

- `handleRetrySuccess` is invoked upon a successful retry attempt. The retry policy chains to a Java action on `retrySuccessAction`.
- `handleFault` is invoked to execute a policy of type `javaAction`.

[Example 12–12](#) shows the data available with `IFaultRecoveryContext`:

Example 12–12 Data Available with `IFaultRecoveryContext`

```
public interface IFaultRecoveryContext {

/**
 * Gets implementation type of the fault.
 * @return
 */
public String getType();

/**
 * @return Get property set of the fault policy action being executed.
 */
public Map getProperties();

/**
 * @return Get fault policy id of the fault policy being executed.
 */
public String getPolicyId();

/**
```

```

    * @return Name of the faulted partner link.
    */
    public String getReferenceName();

    /**
     * @return Port type of the faulted reference .
     */
    public QName getPortType();
}

```

The service engine implementation of this interface provides more information (for example, Oracle BPEL Process Manager). [Example 12–13](#) provides details.

Example 12–13 Service Engine Implementation of *IFaultRecoveryContext*

```

public class BPELFaultRecoveryContextImpl extends BPELXExecLetUtil implements
IBPELFaultRecoveryContext, IFaultRecoveryContext{
...
}

```

Oracle BPEL Process Manager-specific data is available with *IBPELFaultRecoveryContext*, as shown in [Example 12–14](#).

Example 12–14 Oracle BPEL Process Manager-Specific Data

```

public interface IBPELFaultRecoveryContext {
    public void addAuditTrailEntry(String message);

    public void addAuditTrailEntry(String message, Object detail);

    public void addAuditTrailEntry(Throwable t);
    /**
     * @return Get action id of the fault policy action being executed.
     */
    public String getActionId();

    /**
     * @return Type of the faulted activity.
     */
    public String getActivityId();

    /**
     * @return Name of the faulted activity.
     */
    public String getActivityName();

    /**
     * @return Type of the faulted activity.
     */
    public String getActivityType();

    /**
     * @return Correlation id of the faulted activity.
     */
    public String getCorrelationId();

    /**
     * @return BPEL fault that caused the invoke to fault.
     */
    public BPELFault getFault();
}

```

```
/**
 * @return Get index value of the instance
 */
public String getIndex(int i);

/**
 * @return get Instance Id of the current process instance of the faulted
 *         activity.
 */
public long getInstanceId();

/**
 * @return Get priority of the current process instance of the faulted
 *         activity.
 */
public int getPriority();

/**
 * @return Process DN.
 */
public ComponentDN getProcessDN();

/**
 * @return Get status of the current process instance of the faulted
 *         activity.
 */
public String getStatus();

/**
 * @return Get title of the current process instance of the faulted
 *         activity.
 */
public String getTitle();

public Object getVariableData(String name) throws BPELFault;

public Object getVariableData(String name, String partOrQuery)
throws BPELFault;

public Object getVariableData(String name, String part, String query)
throws BPELFault;

/**
 * @param priority
 *         Set priority of the current process instance of the faulted
 *         activity.
 * @return
 */
public void setPriority(int priority);

/**
 * @param status
 *         Set status of the current process instance of the faulted
 *         activity.
 */
public void setStatus(String status);

/**
 * @param title
```

```

*           Set title of the current process instance of the faulted
*           activity.
* @return
*/
public String setTitle(String title);

public void setVariableData(String name, Object value) throws BPELFault;

public void setVariableData(String name, String partOrQuery, Object value)
throws BPELFault;

public void setVariableData(String name, String part, String query,
Object value) throws BPELFault;
}

```

[Example 12–15](#) provides an example of `javaAction` implementation.

Example 12–15 Implementation of a `javaAction`

```

public class TestJavaAction implements IFaultRecoveryJavaClass {
public void handleRetrySuccess(IFaultRecoveryContext ctx) {
System.out.println("This is for retry success");
handleFault(ctx);
}
public String handleFault(IFaultRecoveryContext ctx) {
System.out.println("-----Inside handleFault-----\n" + ctx.toString());

        dumpProperties(ctx.getProperties());
/* Get BPEL specific context here */
BPELFaultRecoveryContextImpl bpelCtx = (BPELFaultRecoveryContextImpl) ctx;
bpelCtx.addAuditTrailEntry("hi there");
System.out.println("Policy Id" + ctx.getPolicyId());
        ...
}

```

12.4.4 What You May Need to Know About Fault Management Behavior When the Number of Instance Retries is Exceeded

When you configure a fault policy to recover instances with the `ora-retry` action and the number of specified instance retries is exceeded, the instance is marked as `open.faulted` (in-flight state). The instance remains active.

Marking instances as `open.faulted` ensures that no instances are lost. You can then configure another fault handling action following the `ora-retry` action in the fault policy file, such as the following:

- Configure an `ora-human-intervention` action to manually perform instance recovery from Oracle Enterprise Manager Fusion Middleware Control Console.
- Configure an `ora-terminate` action to close the instance (mark it as `closed.faulted`) and never retry again.

However, if you do *not* set an action to be performed after an `ora-retry` action in the fault policy file and the number of instance retries is exceeded, the instance *remains* marked as `open.faulted`, and recovery attempts to handle the instance.

For example, if no action is defined in the following fault policy file after `ora-retry`:

```

<Action id="ora-retry">
  <retry>
    <retryCount>2</retryCount>

```

```

        <retryInterval>2</retryInterval>
        <exponentialBackoff/>
    </retry>
</Action>

```

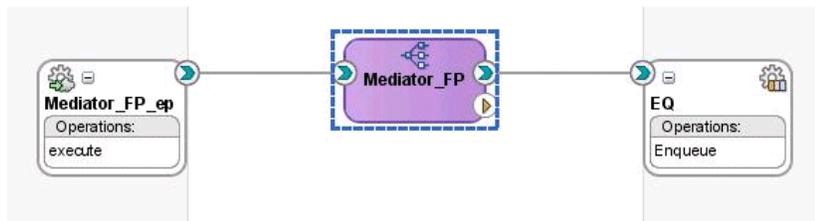
The following actions are performed:

- The invoke activity is attempted (using the above-mentioned fault policy code to handle the fault).
- Two retries are attempted at increasing intervals (after two seconds, then after four seconds).
- If all retry attempts fail, the following actions are performed:
 - A detailed fault error message is logged in the audit trail
 - The instance is marked as `open.faulted` (in-flight state)
 - The instance is picked up and the invoke activity is re-attempted
- Recovery may also fail. In that case, the invoke activity is re-executed. Additional audit messages are logged.

12.4.5 What You May Need to Know About Binding Level Retry Execution Within Fault Policy Retries

If you are testing retry actions on adapters with both JCA-level retries for the outbound direction and a retry action in the fault policy file for outbound failures, the JCA-level (or binding level) retries are executed within the fault policy retries. For example, assume you have designed the application shown in [Figure 12-2](#):

Figure 12-2 SOA Composite Application



You specify the following retry parameters in the `composite.xml` file:

```

<property name="jca.retry.count" type="xs:int" many="false"
  override="may">2</property>
<property name="jca.retry.interval" type="xs:int" many="false"
  override="may">2</property>
<property name="jca.retry.backoff" type="xs:int" many="false"
  override="may">2</property>

```

In the fault policy file for the EQ reference binding component for the outbound direction, you specify the following actions:

```

<retryCount>3</retryCount>
<retryInterval>3</retryInterval>

```

If an outbound failure occurs, the expected behavior is for the JCA retries to occur within the fault policy retries. When the first retry of the fault policy is executed, the

JCA retry is called. In this example, a JCA retry of 2 with an interval of 2 seconds and exponential back off of 2 is executed for every retry of the fault policy:

- Fault policy retry 1:
 - JCA retry 1 (with 2 seconds interval)
 - JCA retry 2 (with 4 seconds interval)
- Fault policy retry 2:
 - JCA retry 1 (with 2 seconds interval)
 - JCA retry 2 (with 4 seconds interval)
- Fault policy retry 3:
 - JCA retry 1 (with 2 seconds interval)
 - JCA retry 2 (with 4 seconds interval)

12.5 Catching BPEL Runtime Faults

BPEL runtime faults can be caught as a named BPEL fault. The `bindingFault` and `remoteFault` can be associated with a message. This action enables the `faultHandler` to get details about the faults.

12.5.1 How to Catch BPEL Runtime Faults

The following procedure shows how to use the provided examples to generate a fault and define a fault handler to catch it. In this case, you modify a WSDL file to generate a fault, and create a catch attribute to catch it.

To catch BPEL runtime faults:

1. Import `RuntimeFault.wsdl` into your process WSDL. `RuntimeFault.wsdl` is seeded into the MDS from `soa.mar` inside `soa-infra-wls.ear` during its deployment.
1. You may see a copy of `soa.mar` in the deployed SOA Infrastructure in the Oracle WebLogic Server domain, which is a JAR/ZIP file containing `RuntimeFault.wsdl`.
2. Declare a variable with `messageType bpelx:RuntimeFaultMessage`.
3. Catch it using the following syntax:

```
<catch faultName="bpelx:remoteFault" | "bpelx:bindingFault"
  faultName="varName">
```

12.6 Getting Fault Details with the `getFaultAsString` XPath Extension Function

The `catchAll` activity is provided to catch possible faults. However, BPEL does not provide a method for obtaining additional information about the captured fault. Use the `getFaultAsString()` XPath extension function to obtain additional information.

12.6.1 How to Get Fault Details with the `getFaultAsString` XPath Extension Function

[Example 12-16](#) shows how to use this function.

Example 12–16 *getFaultAsString() XPath Extension Function*

```
<catchAll>
  <sequence>
    <assign>
      <from expression="bpelx:getFaultAsString()" />
      <to variable="faultVar" part="message" />
    </assign>
    <reply faultName="ns1:myFault" variable="faultVar" .../>
  </sequence>
</catchAll>
```

12.7 Throwing Internal Faults

A BPEL application can generate and receive fault messages. The throw activity has three elements: its name, the name of the fault, and the fault variable. If you add a throw activity to your BPEL process service component, 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.

12.7.1 How to Create a Throw Activity

To create a throw activity:

1. From the Component Palette, drag a **Throw** activity into the designer.
2. Double-click and define the **Throw** activity.
3. Optionally enter a name or accept the default value.
4. To the right of the **Namespace URI** field, click the **Search** icon to select the fault to monitor.
5. Select the fault in the Fault Chooser dialog, and click **OK**.

The namespace URI for the selected fault displays in the **Namespace URI** field. Your fault selection also automatically displays in the **Local Part** field.

[Figure 12–3](#) provides an example of a completed Throw dialog. This example shows the **Throw_Fault_CC_Denied** throw activity of the **Scope_AuthorizeCreditCard** scope activity in the WebLogic Fusion Order Demo application. This activity throws a fault for orders that are not approved.

Figure 12–3 Throw Dialog

6. Click OK.

12.7.2 What Happens When You Create a Throw Activity

[Example 12–17](#) shows the throw activity in the .bpel file after design completion. The `OrderProcessor` process terminates after executing this throw activity.

Example 12–17 Throw Activity

```
<throw name="Throw_Fault_CC_Denied"
      faultName="client:OrderProcessorFault"/>
```

12.8 Returning External Faults

A BPEL process service component 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.

12.8.1 How to Return a Fault in a Synchronous Interaction

The syntax of a reply activity that returns a fault in a synchronous interaction is shown in [Example 12–18](#):

Example 12–18 Reply Activity

```
<reply partnerlink="partner-link-name"
      portType="port-type-name"
      operation="operation-name"
      variable="variable-name" (optional)
      faultName="fault-name">
</reply>
```

Always returning a fault in response to a synchronous request is not very useful. It is better to make the activity part of a conditional branch, in which the first branch is executed if the data requested is available. If the requested data is not available, then the BPEL process service component returns a fault with this information.

For more information, see the following chapters:

- [Chapter 11, "Using Conditional Branching in a BPEL Process"](#) for setting up the conditional structure
- [Chapter 8, "Invoking a Synchronous Web Service from a BPEL Process"](#) for synchronous interactions

12.8.2 How to Return a Fault in an Asynchronous Interaction

In an asynchronous interaction, the client does not wait for a reply. The reply activity is not used to return a fault. Instead, the BPEL process service component returns a fault using a callback operation on the same port type that normally receives the requested information, with an invoke activity.

For more information about asynchronous interactions, see [Chapter 9, "Invoking an Asynchronous Web Service from a BPEL Process."](#)

12.9 Using a Scope Activity to Manage a Group of Activities

A scope activity provides a container and a context for other activities. A scope provides handlers for faults, events, compensation, data variables, and correlation sets. Using a scope activity simplifies a BPEL flow by grouping functional structures. This grouping allows you to collapse them into what appears to be a single element in Oracle BPEL Designer.

[Example 12–19](#) shows a scope named `Scope_FulfillOrder` from the WebLogic Fusion Order Demo application. This scope invokes the `FulfillOrder` mediator component, which determines the shipping method for the order.

Example 12–19 Scope Activity

```
<scope name="Scope_FulfillOrder">
  <variables>
    <variable name="lFulfillOrder_InputVariable"
      messageType="ns17:requestMessage" />
  </variables>
  <sequence>
    <assign name="Assign_OrderData">
      <copy>
        <from variable="gOrderInfoVariable"
          query="/ns4:orderInfoVOSDO" />
        <to variable="lFulfillOrder_InputVariable"
          part="request" query="/ns4:orderInfoVOSDO" />
      </copy>
    </assign>
    <invoke name="Invoke_FulfillOrder"
      inputVariable="lFulfillOrder_InputVariable"
      partnerLink="FulfillOrder.FulfillOrder"
      portType="ns17:execute_ptt" operation="execute" />
  </sequence>
</scope>
```

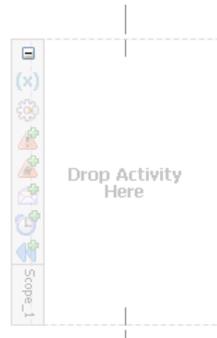
12.9.1 How to Create a Scope Activity

To create a scope activity:

1. From the Component Palette, drag a **Scope** activity into the designer.

2. Open the **scope** activity by double-clicking it or by single-clicking the **Expand** icon.
3. From the Component Palette, drag and define activities to build the functionality within the scope.

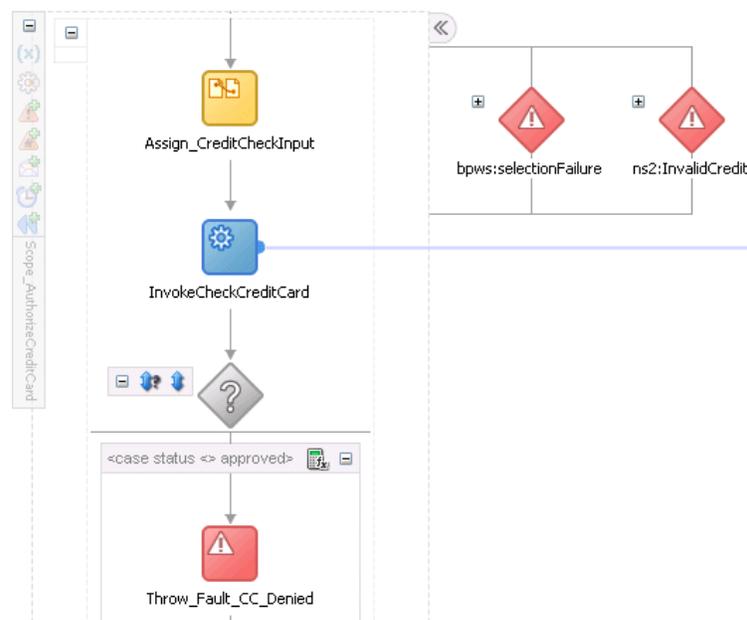
Figure 12–4 Expanded Scope Activity



4. Click **OK**.

When complete, scope activity design can look as shown in [Figure 12–5](#). This example shows the **Scope_AuthorizeCreditCard** scope activity of the WebLogic Fusion Order Demo application.

Figure 12–5 Scope Activity After Design Completion



12.9.2 What Happens After You Create a Scope Activity

[Example 12–20](#) shows the throw activity in the `.bpel` file after design completion. The `Scope_AuthorizeCreditCard` scope activity consists of activities that perform the following actions:

- A catch activity for catching faulted orders in which the credit card number is not provided or the credit type is not valid.

- A throw activity that throws a fault for orders that are not approved.
- An assign activity that takes the credit card type, credit card number, and purchase amount, and assigns it to the input variable for the `CreditCardAuthorizationService` service.
- An invoke activity that calls a `CreditCardAuthorizationService` service to retrieve customer information.
- A switch activity that checks the results of the credit card validation.

Example 12–20 Scope Activity

```

<scope name="Scope_AuthorizeCreditCard">
  <variables>
    <variable name="lCreditCardInput"
      messageType="ns2:CreditAuthorizationRequestMessage"/>
    <variable name="lCreditCardOutput"
      messageType="ns2:CreditAuthorizationResponseMessage"/>
  </variables>
  <faultHandlers>
    <catch faultName="bpws:selectionFailure">
      <sequence>
        <assign name="Assign_noCCNumber">
          <copy>
            <from expression="string('CreditCardCheck - NO
              CreditCard')"/>
            <to variable="gOrderProcessorFaultVariable"
              part="code"/>
          </copy>
        </assign>
        <throw name="Throw_NoCreditCard"
          faultVariable="gOrderProcessorFaultVariable"
          faultName="ns9:OrderProcessingFault"/>
      </sequence>
    </catch>
    <catch faultName="ns2:InvalidCredit">
      <sequence>
        <assign name="Assign_InvalidCreditFault">
          <copy>
            <from expression="concat(bpws:getVariableData
              ('gOrderInfoVariable','/ns4:orderInfoVOSDO/
              ns4:CardTypeCode'), ' is not a valid
              creditcard type')"/>
            <to variable="gOrderProcessorFaultVariable"
              part="summary"/>
          </copy>
          <copy>
            <from expression="string('CreditCardCheck - NOT VALID')"/>
            <to variable="gOrderProcessorFaultVariable"
              part="code"/>
          </copy>
        </assign>
        <throw name="Throw_OrderProcessingFault"
          faultName="ns9:OrderProcessingFault"
          faultVariable="gOrderProcessorFaultVariable"/>
      </sequence>
    </catch>
  </faultHandlers>
  <sequence>
    <assign name="Assign_CreditCheckInput">

```

```

<copy>
  <from variable="gOrderInfoVariable"
    query="/ns4:orderInfoVOSDO/ns4:OrderTotal" />
  <to variable="lCreditCardInput" part="Authorization"
    query="/ns8:AuthInformation/ns8:PurchaseAmount" />
</copy>
<copy>
  <from variable="gOrderInfoVariable"
    query="/ns4:orderInfoVOSDO/ns4:CardTypeCode" />
  <to variable="lCreditCardInput" part="Authorization"
    query="/ns8:AuthInformation/ns8:CCType" />
</copy>
<copy>
  <from variable="gOrderInfoVariable"
    query="/ns4:orderInfoVOSDO/ns4:AccountNumber" />
  <to variable="lCreditCardInput" part="Authorization"
    query="/ns8:AuthInformation/ns8:CCNumber" />
</copy>
</assign>
<invoke name="InvokeCheckCreditCard"
  inputValue="lCreditCardInput"
  outputVariable="lCreditCardOutput"
  partnerLink="CreditCardAuthorizationService"
  portType="ns2:CreditAuthorizationPort"
  operation="AuthorizeCredit" />
<switch name="Switch_EvaluateCCResult">
  <case condition="bpws:getVariableData('lCreditCardOutput','status','
    /ns8:status') != 'APPROVED'">
    <bpelx:annotation>
      <bpelx:pattern>status &lt;&gt; approved</bpelx:pattern>
    </bpelx:annotation>
    <throw name="Throw_Fault_CC_Denied"
      faultName="client:OrderProcessorFault" />
  </case>
</switch>
</sequence>
</scope>

```

12.9.3 What You May Need to Know About Scopes

Scopes can use a significant amount of CPU and memory and should not be overused. Sequence activities use less CPU and memory and can be used to make large BPEL flows more readable.

12.9.4 How to Use a Fault Handler within a Scope

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. To prevent this, contain the parts of the process that have the potential to receive faults within a scope. The scope activity includes the following fault handling capabilities:

- The catch activity works within a scope to catch faults and exceptions before they can throw the entire process into a faulted state. You can use specific fault names in the catch activity to respond in a specific way to an individual fault.
- The catchAll activity catches any faults that are not handled by name-specific catch activities.

[Example 12-21](#) shows the syntax for catch and catch all activities. Assume that a fault named `x:foo` is thrown. The first `catch` is selected if the fault carries no fault data. If

there is fault data associated with the fault, the third `catch` is selected if the type of the fault's data matches the type of variable `bar`. Otherwise, the default `catchAll` handler is selected. Finally, a fault with a fault variable whose type matches the type of `bar` and whose name is not `x:foo` is processed by the second `catch`. All other faults are processed by the default `catchAll` handler.

Example 12–21 Catch and Catch All Activities

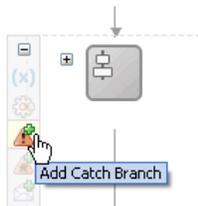
```
<falhandlers>
  <catch faultName="x:foo">
    <empty/>
  </catch>
  <catch faultVariable="bar">
    <empty/>
  </catch>
  <catch faultName="x:foo" faultVariable="bar">
    <empty/>
  </catch>
  <catchAll>
    <empty/>
  </catchAll>
</falhandlers>
```

12.9.5 How to Create a Catch Activity

To create a catch activity:

1. In the expanded **Scope** activity, click **Add Catch Branch**.

Figure 12–6 Add Catch Branch

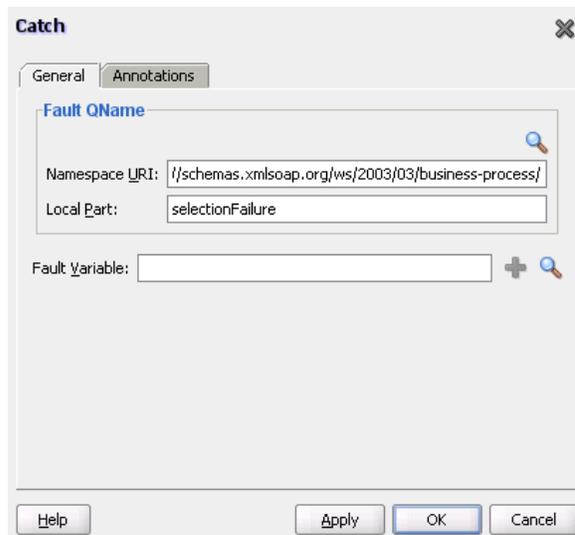


This creates a catch activity in the right side of the scope activity.

2. Double-click the **Catch** activity.
3. Optionally enter a name.
4. To the right of the **Namespace URI** field, click the **Search** icon to select the fault.
5. Select the fault in the Fault Chooser dialog, and click **OK**.

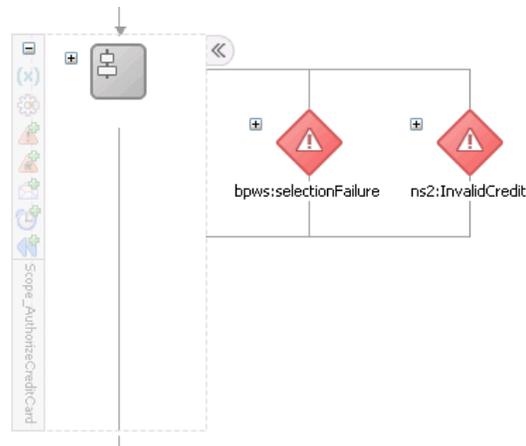
The namespace URI for the selected fault displays in the **Namespace URI** field. Your fault selection also automatically displays in the **Local Part** field.

Figure 12–7 provides an example of a Catch dialog. This example shows the **selectionFailure** catch activity of the **Scope_AuthorizeCreditCard** scope activity in the WebLogic Fusion Order Demo application. This catch activity catches orders in which the credit card number is not provided.

Figure 12–7 Catch Dialog

6. Design additional fault handling functionality.
7. Click OK.

Figure 12–8 provides an example of two catch activities for the **Scope_AuthorizeCreditCard** scope activity. The second catch activity catches credit types that are not valid.

Figure 12–8 Catch Activities in the Designer

12.9.6 What Happens When You Create a Catch Branch

Figure 12–22 shows the catch activity in the .bpel file after design completion. The `selectionFailure` catch activity catches orders in which the credit card number is not provided and the `InvalidCredit` catch activity catches credit types that are not valid.

Example 12–22 Catch Branch

```
<faultHandlers>
  <catch faultName="bpws:selectionFailure">
    <sequence>
```

```

        <assign name="Assign_noCCNumber">
            <copy>
                <from expression="string('CreditCardCheck - NO CreditCard')"/>
                <to variable="gOrderProcessorFaultVariable"
                    part="code"/>
            </copy>
        </assign>
        <throw name="Throw_NoCreditCard"
            faultVariable="gOrderProcessorFaultVariable"
            faultName="ns9:OrderProcessingFault"/>
    </sequence>
</catch>
<catch faultName="ns2:InvalidCredit">
    <sequence>
        <assign name="Assign_InvalidCreditFault">
            <copy>
                <from expression="concat(bpws:getVariableData
                    ('gOrderInfoVariable','/ns4:orderInfoVOSDO/ns4:CardTypeCode'),'
                    is not a valid creditcard type')"/>
                <to variable="gOrderProcessorFaultVariable"
                    part="summary"/>
            </copy>
            <copy>
                <from expression="string('CreditCardCheck - NOT VALID')"/>
                <to variable="gOrderProcessorFaultVariable"
                    part="code"/>
            </copy>
        </assign>
        <throw name="Throw_OrderProcessingFault"
            faultName="ns9:OrderProcessingFault"
            faultVariable="gOrderProcessorFaultVariable"/>
    </sequence>
</catch>
</faultHandlers>

```

12.9.7 How to Create an Empty Activity to Insert No-Op Instructions into a Business Process

There is often a need to use an activity that does nothing. An example is when a fault must be caught and suppressed. In this case, you can use the empty activity to insert a no-op instruction into a business process.

To create an empty activity:

1. From the Component Palette, drag an **Empty** activity into the designer.
2. Double-click the **Empty** activity.

The Empty dialog appears, as shown in [Figure 12-9](#).

Figure 12–9 Empty Activity

3. Optionally enter a name.
4. Click OK.

12.9.8 What Happens When You Create an Empty Activity

The syntax for an `empty` activity is shown in [Example 12–23](#).

Example 12–23 Empty Activity

```
<empty standard-attributes>
  standard-elements
</empty>
```

If no `catch` or `catchAll` is selected, the fault is not caught by the current scope and is rethrown to the immediately enclosing scope. If the fault occurs in (or is rethrown to) the global process scope, and there is no matching fault handler for the fault at the global level, the process terminates abnormally. This is as though a terminate activity (described in [Section 12.11, "Using the Terminate Activity to Stop a Business Process Instance"](#)) had been performed.

12.10 Using Compensation After Undoing a Series of Operations

Compensation occurs when the BPEL process service component cannot complete a series of operations after some have completed, and the BPEL process service component must backtrack and undo the previously completed transactions. For example, if a BPEL process service component is designed to book a rental car, a hotel, and a flight, it may book the car and the hotel and then be unable to book a flight for the right day. In this case, the BPEL flow performs compensation by going back and unbooking the car and the hotel.

12.10.1 How to Use Compensation After Undoing a Series of Operations

You can invoke a compensation handler by using the `compensate` activity, which names the scope for which the compensation is to be performed (that is, the scope whose compensation handler is to be invoked). A compensation handler for a scope is available for invocation only when the scope completes normally. Invoking a

compensation handler that has not been installed is equivalent to using the empty activity (it is a no-op). This ensures that fault handlers do not have to rely on state to determine which nested scopes have completed successfully. The semantics of a process in which an installed compensation handler is invoked multiple times are undefined.

The ability to explicitly invoke the compensate activity is the underpinning of the application-controlled error-handling framework of *Business Process Execution Language for Web Services Specification*. You can use this activity only in the following parts of a business process:

- In a fault handler of the scope that immediately encloses the scope for which compensation is to be performed.
- In the compensation handler of the scope that immediately encloses the scope for which compensation is to be performed.

For example:

```
<compensate scope="RecordPayment" />
```

If a scope being compensated by name was nested in a loop, the BPEL process service component invokes the instances of the compensation handlers in the successive iterations in reverse order.

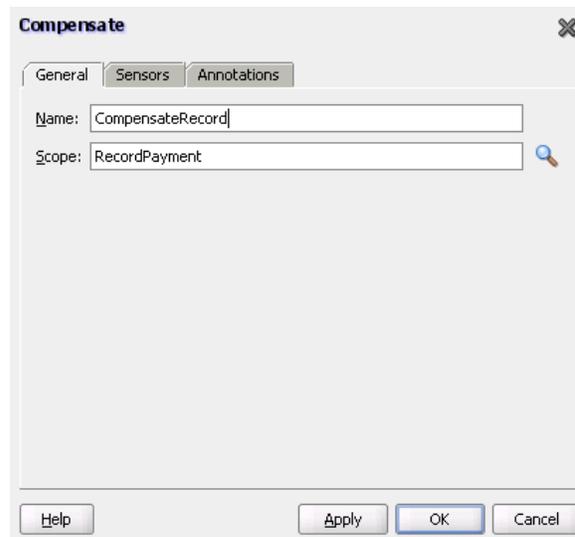
If the compensation handler for a scope is absent, the default compensation handler invokes the compensation handlers for the immediately enclosed scopes in the reverse order of the completion of those scopes.

The compensate form, in which the scope name is omitted in a compensate activity, explicitly invokes this default behavior. This is useful when an enclosing fault or compensation handler must perform additional work, such as updating variables or sending external notifications, in addition to performing default compensation for inner scopes. The compensate activity in a fault or compensation handler attached to the outer scope invokes the default order of compensation handlers for completed scopes directly nested within the outer scope. You can mix this activity with any other user-specified behavior except for the explicit invocation of the nested scope within the outer scope. Explicitly invoking a compensation for such a scope nested within the outer scope disables the availability of default-order compensation.

12.10.2 How to Create a Compensate Activity

To create a compensate activity:

1. From the Component Palette, drag an **Compensate** activity into the designer.
2. Double-click the **Compensate** activity.
3. Select a scope activity in which to invoke the compensation handler.

Figure 12–10 Compensate Activity

4. Click OK.

12.10.3 What Happens When You Create a Compensate Activity

If an invoke activity has a compensation handler defined inline, then the name of the activity is the name of the scope to be used in the compensate activity. The syntax is shown in [Example 12–24](#):

Example 12–24 Compensation Handler

```
<compensate scope="ncname"? standard-attributes>
  standard-elements
</compensate>
```

12.11 Using the Terminate Activity to Stop a Business Process Instance

The terminate activity immediately terminates the behavior of a business process instance within which the terminate activity is performed. All currently running activities must be terminated as soon as possible without any fault handling or compensation behavior. The terminate activity does not send any notifications of the status of a BPEL process service component. If you are going to use the terminate activity, first program notifications to the interested parties.

12.11.1 How to Create a Terminate Activity

To create a terminate activity:

1. From the Component Palette in Oracle JDeveloper, drag a **Terminate** activity into the designer. [Figure 12–11](#) provides an example.

Figure 12–11 Terminate Activity



2. Double-click the **terminate** activity.
3. Optionally enter a name.
4. Click **OK**.

12.11.2 What Happens When You Create a Terminate Activity

The syntax for the `terminate` activity is shown in [Example 12–25](#). This stops the business process instance.

Example 12–25 Terminate Activity

```
<terminate standard-attributes>  
  standard-elements  
</terminate>
```

Incorporating Java and Java EE Code in a BPEL Process

This chapter describes how to incorporate sections of Java code into BPEL process service components in SOA composite applications.

This chapter includes the following sections:

- [Section 13.1, "Introduction to Java and Java EE Code in BPEL Processes"](#)
- [Section 13.2, "Incorporating Java and Java EE Code in BPEL Processes"](#)
- [Section 13.3, "Adding Custom Classes and JAR Files"](#)
- [Section 13.4, "Using Java Embedding in a BPEL Process in Oracle JDeveloper"](#)
- [Section 13.5, "Embedding Service Data Objects with `bpelx:exec`"](#)

13.1 Introduction to Java and Java EE Code in BPEL Processes

This chapter explains how to incorporate sections of Java code into a BPEL process. This is particularly useful when there is Enterprise JavaBeans Java code that can perform the necessary function, and you want to use the existing code rather than start over with BPEL.

13.2 Incorporating Java and Java EE Code in BPEL Processes

There are several methods for incorporating Java and Java EE code in BPEL processes:

- Wrap as a Simple Object Access Protocol (SOAP) service
- Embed Java code snippets into a BPEL process with the `bpelx:exec` tag
- Use an XML facade to simplify DOM manipulation
- Use `bpelx:exec` built-in methods
- Use Java code wrapped in a service interface

13.2.1 How to Wrap Java Code as a SOAP Service

You can wrap the Java code as a Simple Object Access Protocol (SOAP) service. This method requires that the Java application have a BPEL-compatible interface. A Java application wrapped as a SOAP service appears as any other web service, which can be used by many different kinds of applications. There are also tools available for writing SOAP wrappers.

13.2.2 What You May Need to Know About Wrapping Java Code as a SOAP Service

A Java application wrapped as a SOAP service has the following drawbacks:

- There may be reduced performance due to the nature of converting between Java and SOAP, and back and forth.
- Since SOAP inherently has no support for transactions, this method loses atomic transactionality, that is, the ability to perform several operations in an all-or-none mode (such as debiting one bank account while crediting another, where either both transactions must be completed, or neither of them).

13.2.3 How to Embed Java Code Snippets into a BPEL Process with the `bpelx:exec` Tag

You can embed Java code snippets directly into the BPEL process using the Java BPEL `exec` extension `bpelx:exec`. The benefits of this approach are speed and transactionality. It is recommended that you incorporate only small segments of code. BPEL is about separation of business logic from implementation. If you remove a lot of Java code in your process, you lose that separation. Java embedding is recommended for short utility-like operations, rather than business code. Place the business logic elsewhere and call it from BPEL.

The server executes any snippet of Java code contained within a `bpelx:exec` activity, within its Java Transaction API (JTA) transaction context.

The BPEL tag `bpelx:exec` converts Java exceptions into BPEL faults and then adds them into the BPEL process.

The Java snippet can propagate its JTA transaction to session and entity beans that it calls.

For example, a `SessionBeanSample.bpel` file uses the `bpelx:exec` tag shown in [Example 13-1](#) to embed the `invokeSessionBean` Java bean:

Example 13-1 `bpelx:exec` Tag

```
<bpelx:exec name="invokeSessionBean" language="java" version="1.5">
  <![CDATA[
    try {
      Object homeObj = lookup("ejb/session/CreditRating");
      Class cls = Class.forName(
        "com.otn.samples.sessionbean.CreditRatingServiceHome");
      CreditRatingServiceHome ratingHome = (CreditRatingServiceHome)
        PortableRemoteObject.narrow(homeObj,cls);
      if (ratingHome == null) {
        addAuditTrailEntry("Failed to lookup 'ejb.session.CreditRating'"
          + ". Ensure that the bean has been"
          + " successfully deployed");
      }
      return;
    }
    CreditRatingService ratingService = ratingHome.create();

    // Retrieve ssn from scope
    Element ssn =
      (Element)getVariableData("input", "payload", "/ssn");

    int rating = ratingService.getRating( ssn.getNodeValue() );
    addAuditTrailEntry("Rating is: " + rating);

    setVariableData("output", "payload",
      "/tns:rating", new Integer(rating));
  ]]>
</bpelx:exec>
```

```

    } catch (NamingException ne) {
        addAuditTrailEntry(ne);
    } catch (ClassNotFoundException cnfe) {
        addAuditTrailEntry(cnfe);
    } catch (CreateException ce) {
        addAuditTrailEntry(ce);
    } catch (RemoteException re) {
        addAuditTrailEntry(re);
    }
  ]]>
</bpelx:exec>

```

13.2.4 How to Use an XML Facade to Simplify DOM Manipulation

You can use an XML facade to simplify DOM manipulation. Oracle BPEL Process Manager provides a lightweight Java Architecture for XML Binding (JAXB)-like Java object model on top of XML (called a facade). An XML facade provides a Java bean-like front end for an XML document or element that has a schema. Facade classes can provide easy manipulation of the XML document and element in Java programs.

You add the XML facade by using a `createFacade` method within the `bpelx:exec` statement in the `.bpel` file. [Example 13–2](#) provides an example:

Example 13–2 Addition of XML facade

```

<bpelx:exec name= ...
  <![CDATA
    ...
    Element element = ...
    (Element)getVariableData("input","payload","/loanApplication/"):
    //Create an XMLFacade for the Loan Application Document
    LoanApplication xmlLoanApp=
      LoanApplicationFactory.createFacade(element);
    ...
  ]]>

```

13.2.5 How to Use `bpelx:exec` Built-in Methods

[Table 13–1](#) lists a set of `bpelx:exec` built-in methods that you can use to read and update scope variables, instance metadata, and audit trails.

Table 13–1 Built in Methods for `bpelx:exec`

Method Name	Description
Object lookup(String name)	JNDI access
Locator getLocator()	Oracle BPEL Process Manager locator
long getInstanceId()	Unique ID associated with each instance
String setTitle(String title) / String getTitle()	Title of this instance
String setStatus(String status) / String getStatus()	Status of this instance
void setCompositeInstanceTitle(String title)	Set the composite instance title
void setIndex(int i, String value) / String getIndex(int i)	Six indexes can be used for search

Table 13–1 (Cont.) Built in Methods for *bpelx:exec*

Method Name	Description
<code>void setPriority(int priority) / int getPriority()</code>	Priority
<code>void setCreator(String creator) / String getCreator()</code>	Who initiated this instance
<code>void setCustomKey(String customKey) / String getCustomKey()</code>	Second primary key
<code>void setMetadata(String metadata) / String getMetadata ()</code>	Metadata for generating lists
<code>String getPreference(String key)</code>	Access preference
<code>void addAuditTrailEntry(String message, Object detail)</code>	Add an entry to the audit trail
<code>void addAuditTrailEntry(Throwable t)</code>	Access file stored in the suitcase
<code>Object getVariableData(String name) throws BPELFault</code>	Access and update variables stored in the scope
<code>Object getVariableData(String name, String partOrQuery) throws BPELFault</code>	Access and update variables.
<code>Object getVariableData(String name, String part, String query)</code>	Access and update variables.
<code>void setVariableData(String name, Object value)</code>	Set variable data.
<code>void setVariableData(String name, String part, Object value)</code>	Set variable data.
<code>void setVariableData(String name, String part, String query, Object value)</code>	Set variable data.

13.2.6 How to Use Java Code Wrapped in a Service Interface

Not all applications expose a service interface. You may have a scenario in which a business process must use custom Java code. For this scenario, you can:

- Write custom Java code.
- Create a service interface in which to embed the code.
- Invoke the Java code as a web service over SOAP.

For example, assume you create a BPEL process service component in a SOA composite application that invokes a service interface through a SOAP reference binding component. For this example, the service interface used is an Oracle Application Development Framework (ADF) Business Component.

The high-level instructions for this scenario are as follows.

To use Java code wrapped in a service interface:

1. Create an Oracle ADF Business Component service in Oracle JDeveloper.
This action generates a WSDL file and XSD file for the service.
2. Create a SOA application that includes a BPEL process service component. Ensure that the BPEL process service component is exposed as a composite service. This

automatically connects the BPEL process to an inbound SOAP service binding component.

3. Import the Oracle ADF Business Component service WSDL into the SOA composite application.
4. Create a web service binding to the Oracle ADF Business Component service interface.
5. Design a BPEL process in which you perform the following tasks:
 - a. Create a partner link for the Oracle ADF Business Component service `portType`.
 - b. Create an assign activity. For this example, this step copies data (for example, a static XML fragment) into a variable that is passed to the Oracle ADF Business Component service.
 - c. Create an invoke activity and connect to the partner link you created in Step 5a.
6. Connect (wire) the partner link reference to the composite reference binding component. This reference uses a web service binding to enable the Oracle ADF Business Component service to be remotely deployed.
7. Deploy the SOA application.
8. Invoke the SOA application from the Test Web Service page in Oracle Enterprise Manager Fusion Middleware Control Console.

For more information on creating Oracle ADF Business Components, see *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

For more information on invoking a SOA composite application, see *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite*.

13.3 Adding Custom Classes and JAR Files

You can add custom classes and JAR files to an SOA composite application. A SOA extension library for adding extension classes and JARs to an SOA composite application is available in the `$ORACLE_HOME/soa/modules/oracle.soa.ext_11.1.1` directory. For Oracle JDeveloper, custom classes and JARs are added to the `application_name/project/sca-inf/lib` directory.

13.3.1 How to Add Custom Classes and JAR Files

If the classes are used in `bpelx:exec`, you must also add the JARs in `bpelcClasspath` in `bpel-config.xml`. In addition, ensure that the JARs are loaded by SOA composite application.

To add custom classes:

1. Copy the classes to the `classes` directory.
2. Restart Oracle WebLogic Server.

To add custom JARs:

1. Copy the JAR files to this directory or its subdirectory.
2. Run `ant`.
3. Restart Oracle WebLogic Server.

13.4 Using Java Embedding in a BPEL Process in Oracle JDeveloper

In Oracle JDeveloper, you can add the `bpelx:exec` activity and copy the code snippet into a dialog box.

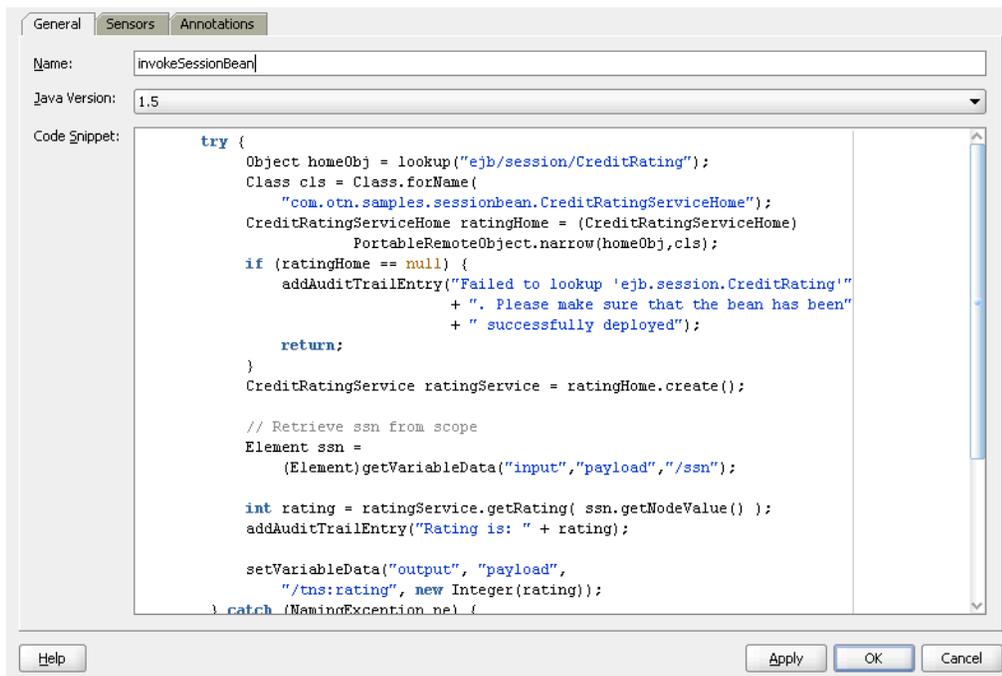
Note: For custom classes, you must include any JAR files required for embedded Java code in the `bpelcClasspath` variable in the `ORACLE_HOME/domains/user_domain_name/config/soa-infra/configuration/bpel-config.xml` file. The JAR files are then added to the class path of the BPEL loader. If multiple JAR files are included, they must be separated by a colon (:) on UNIX and a semicolon (;) on Windows.

13.4.1 How To Use Java Embedding in a BPEL Process in Oracle JDeveloper

To use Java embedding in a BPEL process in Oracle JDeveloper:

1. From the Component Palette, drag the **Java Embedding** activity into the designer.
2. Double-click the **Java Embedding** activity to display the Java Embedding dialog.
3. In the **Name** field, enter a name.
4. In the **Code Snippet** field, enter (or cut and paste) the Java code.

Figure 13–1 *bpel:exec Code Example*



Note: As an alternative to writing Java code in the Java Embedding activity, you can place your Java code in a JAR file, put it in the class path, and call your methods from within the Java Embedding activity.

13.5 Embedding Service Data Objects with bpel:exec

You can embed service data object (SDO) code in the .bpel file with the `bpel:exec` tag. In the syntax provided in [Example 13-3](#), `mytest.apps.SDOHelper` is a Java class that modifies SDOs.

Example 13-3 Embedding SDO Objects with the bpel:exec tag

```
</bpel:exec>
<bpel:exec name="ModifyInternalSDO" version="1.5" language="java">
  <![CDATA[try{
    Object o = getVariableData("VarSDO");
    Object out = getVariableData("ExtSDO");
    System.out.println("BPEL:Modify VarSDO... " + o + " ExtSDO: " + out);
    mytest.apps.SDOHelper.print(o);
    mytest.apps.SDOHelper.print(out);
    mytest.apps.SDOHelper.modifySDO(o);
    System.out.println("BPEL:After Modify VarSDO... " + o + " ExtSDO: " + out);
    mytest.apps.SDOHelper.print(o);
    mytest.apps.SDOHelper.print(out);
  }catch(Exception e)
  {
    e.printStackTrace();
  }
}]]>
</bpel:exec>
```

[Example 13-4](#) provides an example of the Java classes `modifySDO(o)` and `print(o)` that are embedded in the BPEL file.

Example 13-4 Java Classes

```
public static void modifySDO(Object o){
    if(o instanceof commonj.sdo.DataObject)
    {
        ((DataObject)o).getChangeSummary().beginLogging();
        SDOType type = (SDOType)((DataObject)o).getType();
        HelperContext hCtx = type.getHelperContext();
        List<DataObject> lines =
            (List<DataObject>)((DataObject)o).get("line");
        for (DataObject line: lines) {
            line.set("eligibilityStatus", "Y");
        }
    } else {
        System.out.println("SDOHelper.modifySDO(): " + o + " is not a
            DataObject!");
    }
}
. . .
. . .
public static void print(Object o)    {
    try{
        if(o instanceof commonj.sdo.DataObject)
        {
            DataObject sdo = (commonj.sdo.DataObject)o;
            SDOType type = (SDOType) sdo.getType();
            HelperContext hCtx = type.getHelperContext();
            System.out.println(hCtx.getXMLHelper().save(sdo, type.getURI(),
                type.getName()));
        } else {
            System.out.println("SDOHelper.print(): Not a sdo " + o);
        }
    }
}
```

```
    }  
  }catch(Exception e)  
  {  
    e.printStackTrace();  
  }  
}
```

Using Events and Timeouts in BPEL Processes

This chapter describes how to use events and timeouts. Because web services can take a long time to return a response, a BPEL process service component must be able to time out and continue with the rest of the flow after a period of time.

This chapter includes the following sections:

- [Section 14.1, "Introduction to Event and Timeout Concepts"](#)
- [Section 14.2, "Creating a Pick Activity to Select Between Continuing a Process or Waiting"](#)
- [Section 14.3, "Creating a Wait Activity to Set an Expiration Time"](#)
- [Section 14.4, "Setting Timeouts for Synchronous Processes"](#)

14.1 Introduction to Event and Timeout Concepts

This chapter provides an example of how to program a BPEL process service component to wait one minute for a response from a web service named Star Loan that provides loan offers. If Star Loan does not respond in one minute, then the BPEL process service component automatically selects an offer from another web service named United Loan. In the real world, the time limit is more like 48 hours. However, for this example, you do not want to wait that long to see if your BPEL process service component is working properly.

Because asynchronous web services can take a long time to return a response, a BPEL process service component must be able to time out, or give up waiting, and continue with the rest of the flow after a certain amount of time. You can use the pick activity to configure a BPEL flow to either wait a specified amount of time or to continue performing its duties. To set an expiration period for the time, you can use the wait activity.

14.2 Creating a Pick Activity to Select Between Continuing a Process or Waiting

The pick activity provides two branches, each one with a condition. The branch that has its condition satisfied first is executed. In the following example, one branch's condition is to receive a loan offer, and the other branch's condition is to wait a specified amount of time.

[Figure 14-1](#) provides an overview. The following activities take place (in order of priority):

1. An invoke activity initiates a service, in this case, a request for a loan offer from Star Loan.
2. The pick activity begins next. It has the following conditions:
 - onMessage

This condition has code for receiving a reply in the form of a loan offer from the Star Loan web service. The onMessage code is equal to the code for receiving a response from the Star Loan web service before a timeout was added.
 - onAlarm

This condition has code for a timeout of one minute. This time is defined as PT1M, which means to wait one minute before timing out. In this timeout setting:

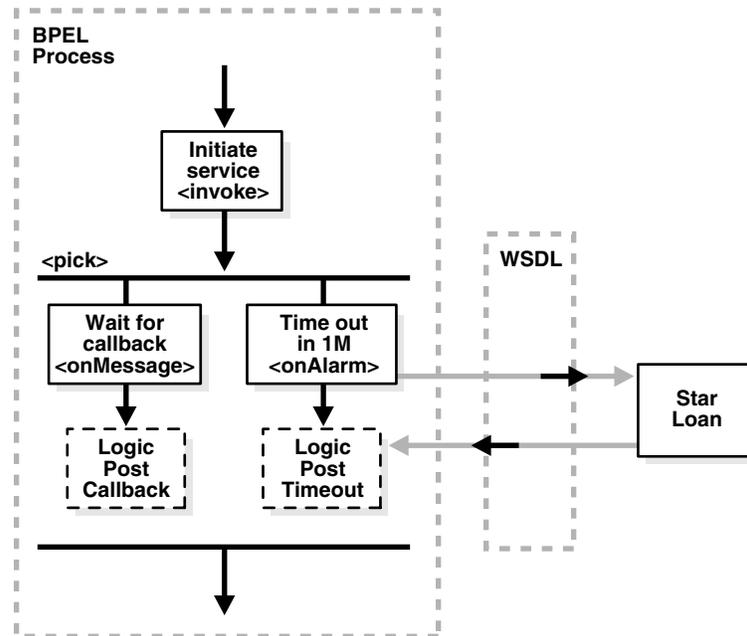
 - S stands for seconds
 - M for one minute
 - H for hour
 - D for day
 - Y for year

In the unlikely event that you want a time limit of 1 year, 3 days, and 15 seconds, you enter it as PT1Y3D15S. The remainder of the code sets the loan variables selected and approved to `false`, sets the annual percentage rate (APR) at 0.0, and copies this information into the `loanOffer` variable.

The time duration format is specified by the BPEL standard. For more detailed information on the time duration format, see the duration section of the most current *XML Schema Part 2: Datatypes* document at:

<http://www.w3.org/TR/xmlschema-2/#duration>
3. The pick activity condition that completes first is the one that the BPEL process service component executes. The other branch then is not executed.

Figure 14–1 Overview of the Pick Activity



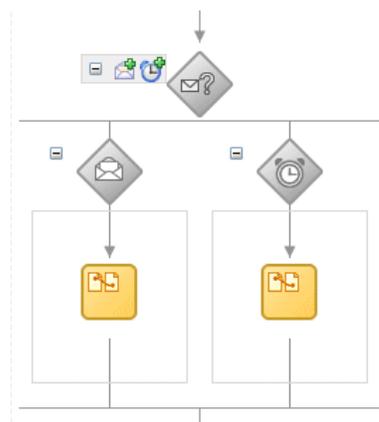
14.2.1 How To Create a Pick Activity

To create a pick activity:

1. In the SOA Composite Editor, double-click the BPEL process service component.
2. From the Component Palette, drag a **Pick** activity into the designer.
3. Expand the **Pick** activity.

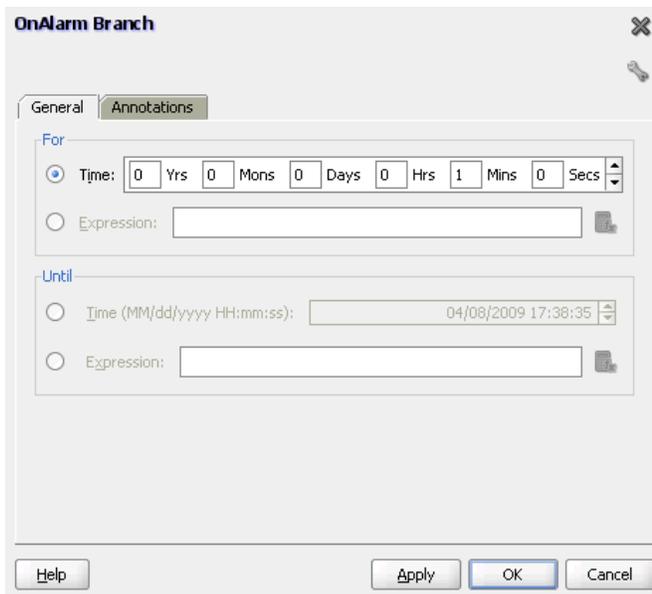
The **Pick** activity includes the **onMessage** (envelope icon) and **onAlarm** (alarm clock icon) branches. Figure 14–2 provides an example.

Figure 14–2 Pick Activity



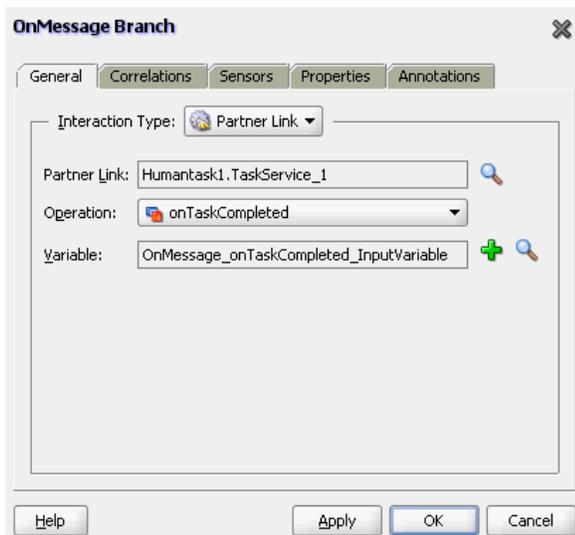
4. Double-click the **OnAlarm** branch of the **pick** activity and set its time limit to 1 minute instead of 1 hour. Figure 14–3 provides an example.

Figure 14–3 OnAlarm Branch



5. Click **OK**.
6. Double-click the **onMessage** branch. [Figure 14–4](#) provides an example.

Figure 14–4 onMessage Branch



7. Edit its attributes to receive the response from the loan service.

14.2.2 What Happens When You Create a Pick Activity

The code segment in [Example 14–1](#) defines the pick activity for this operation after design completion:

Example 14–1 Pick Activity

```
<pick>
  <!-- receive the result of the remote process -->
```

```

<onMessage partnerLink="LoanService"
  portType="services:LoanServiceCallback"
  operation="onResult" variable="loanOffer">

  <assign>
  <copy>
    <from variable="loanOffer" part="payload"/>
    <to variable="output" part="payload"/>
  </copy>
  </assign>

</onMessage>
<!-- wait for one minute, then timeout -->
<onAlarm for="PT1M">
  <assign>
    <copy>
      <from>
        <loanOffer xmlns="http://www.autoloan.com/ns/autoloan">
          <providerName>Expired</providerName>
          <selected type="boolean">false</selected>
          <approved type="boolean">false</approved>
          <APR type="double">0.0</APR>
        </loanOffer>
      </from>
      <to variable="loanOffer" part="payload"/>
    </copy>
  </assign>
</onAlarm>
</pick>

```

14.3 Creating a Wait Activity to Set an Expiration Time

The wait activity allows a process to wait for a given time period or until a time limit has been reached. Exactly one of the expiration criteria must be specified. A typical use of this activity is to invoke an operation at a certain time. You typically enter an expression that is dependent on the state of a process.

When specifying a time period for waiting, note the following:

- Wait times cannot be guaranteed if they are scheduled with other events that require processing. Due to this additional processing, the actual wait time can be greater than the wait time specified in the BPEL process.
- Wait times of less than two seconds are ignored by the server. Wait times above two seconds, but less than one minute, may not get executed in the exact, specified time. However, wait times in minutes do execute in the specified time.
- The default value of 2 seconds for wait times is specified with the `minBPELWait` property in the `bpel-config.xml` file. You can set this property to any value and the wait delay is bypassed for any waits less than `minBPELWait`.

Note: Quartz version 1.6 is supported for scheduling expiration events on wait activities.

14.3.1 How To Create a Wait Activity

To create a wait activity:

1. From the Component Palette, drag a **Wait** activity into the designer.
2. Double-click the **Wait** activity to display the Wait dialog.
3. In the **For** section, enter the amount of time for which to wait.
4. In the **Until** section, select the deadline for which to wait, as shown in [Figure 14-5](#).

Figure 14-5 Wait Dialog

14.3.2 What Happens When You Create a Wait Activity

Exactly one of the expiration criteria must be specified, as shown in [Example 14-2](#).

Example 14-2 Wait Activity

```
<wait (for="duration-expr" | until="deadline-expr") standard-attributes>
  standard-elements
</wait>
```

14.4 Setting Timeouts for Synchronous Processes

For synchronous processes that connect to a remote database, you must increase the `syncMaxWaitTime` timeout property.

14.4.1 How To Set Timeouts for Synchronous Processes

To set timeouts for synchronous processes:

1. Open the `ORACLE_HOME/domains/user_domain_name/config/soa-infra/configuration/bpel-config.xml` file.

2. Edit the value for the `syncMaxWaitTime` property. [Example 14–3](#) provides an example.

Example 14–3 `syncMaxWaitTime` timeout property

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<bpel-config xmlns="http://xmlns.oracle.com/soa/config/bpel" version="11.1.0">
  <!--bpelcClasspath>custom_bpelc_classpath</bpelcClasspath-->
  <dspSystemThreads>2</dspSystemThreads>
  <dspInvokeThreads>20</dspInvokeThreads>
  <dspEngineThreads>30</dspEngineThreads>
  <dspMaxRequestDepth>600</dspMaxRequestDepth>
  <auditLevel>inherit</auditLevel>
  <oneWayDeliveryPolicy>on</oneWayDeliveryPolicy>
  <statsLastN>-1</statsLastN>
  <auditDetailThreshold>50000</auditDetailThreshold>
  <largeDocumentThreshold>100000</largeDocumentThreshold>
  <validateXML>>false</validateXML>
  <expirationMaxRetry>5</expirationMaxRetry>
  <expirationRetryDelay>120</expirationRetryDelay>
  <qualityOfService>DirectWrite</qualityOfService>
  <syncMaxWaitTime>45</syncMaxWaitTime>
  <instanceKeyBlockSize>10000</instanceKeyBlockSize>
  <maximumNumberOfInvokeMessagesInCache>100000
    </maximumNumberOfInvokeMessagesInCache>
</bpel-config>
```

Coordinating Master and Detail Processes

This chapter describes how to coordinate master and detail processes in a BPEL process. This coordination enables you to specify the tasks performed by a master BPEL process and its related detail BPEL processes. This is sometimes referred to as a parent and child relationship.

This chapter includes the following sections:

- [Section 15.1, "Introduction to Master and Detail Process Coordinations"](#)
- [Section 15.2, "Defining Master and Detail Process Coordination in Oracle JDeveloper"](#)

15.1 Introduction to Master and Detail Process Coordinations

Master and detail coordinations consist of a one-to-many relationship between a single master process and multiple detail processes.

For example, assume a business process imports sales orders into an application. Each sales order consists of a header (customer information, ship-to address, and so on) and multiple lines (item name, item number, item quantity, price, and so on).

The following tasks are performed to execute the order:

- Validate the header. If the header is invalid, processing stops.
- Validate each line. If any lines are invalid, they are marked as invalid and processing stops.
- Perform inventory checks for each item. If an item is not available, a work order is created to assemble it.
- Stage items at the shipping dock after items for each line are available.
- Ship the order to the customer.

To perform these tasks, create a master process to check and validate each header and multiple BPEL processes to check and validate each line item.

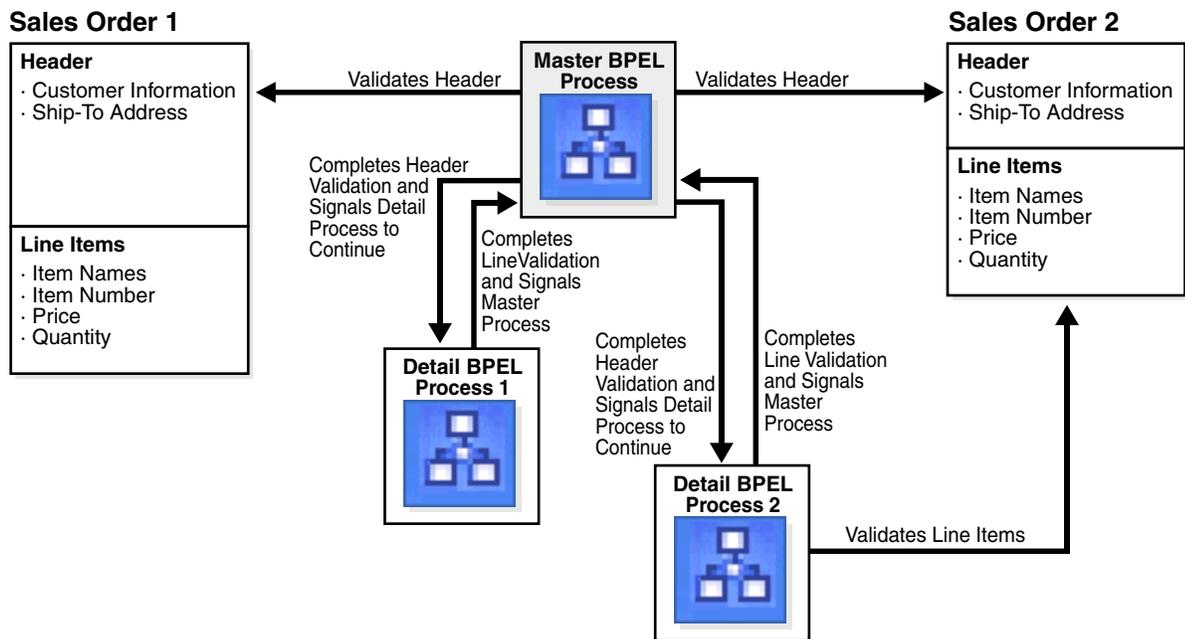
Potential coordination points are as follows:

- The master process must signal the detail processes that header validation is successful and to continue processing.
- Each detail process must signal the master process after line item validation is complete.
- Each detail process must signal the master process after the line item is available in inventory.

- After all line items are available, the master must signal each detail process to move its line item to the shipping dock (the dock may become too crowded if items are simply moved as soon as they are available).
- After all lines have been moved, the master process must execute logic to ship the fulfilled order to the customer.

Figure 15–1 provides an overview of the header and line item validation coordination points between one master process and two detail processes.

Figure 15–1 Master and Detail Coordination Overview (One BPEL Process to Two Detail Processes)



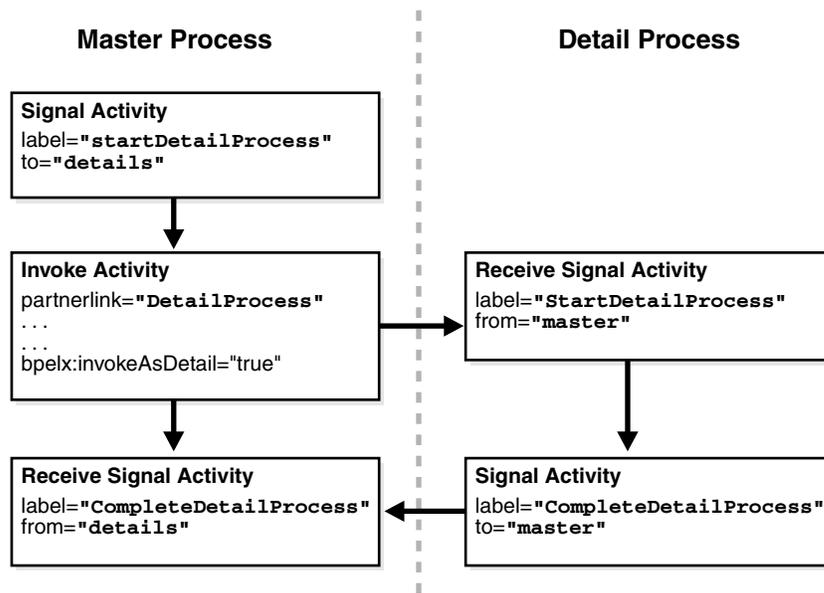
The following BPEL process activities coordinate actions between the master and detail processes:

- signal: notifies the other processes (master or detail) to continue processing
- receive signal: waits until it receives the proper notification signal from the other process (master or detail) before continuing its processing

Both activities are coordinated with label attributes defined in the BPEL process files. Labels are declared per master process definition.

Figure 15–2 provides an overview of the BPEL process flow coordination.

Figure 15–2 Master and Detail Syntax Overview (One BPEL Process to One Detail Process)



As shown in Figure 15–2, each master and detail process includes a signal and receive signal activity. Table 15–1 describes activity responsibilities based on the type of process in which they are defined.

Table 15–1 Master and Detail Process Coordination Responsibilities

If A...	Contains A...	Then...
Master process	Signal activity	The master process signals all of its associated detail processes at runtime.
Detail process	Receive signal activity	The detail process waits until it receives the signal executed by its master process.
Detail process	Signal activity	The detail process signals its associated master process at runtime that processing is complete.
Master process	Receive signal activity	The master process waits until it receives the signal executed by all of its detail processes.

If the signal activity executes before the receive signal activity, the state set by the signal activity is persisted and still effective for a later receive signal activity to read.

15.1.1 BPEL File Definition for the Master Process

The BPEL file for the master process defines coordination with the detail processes. The BPEL file shows that the master process interacts with the partner links of several detail processes. Example 15–1 provides an example.

Example 15–1 BPEL File Definition for the Master Process

```

<process name="MasterProcess"
. . .
. . .
  <partnerLinks>
    <partnerLink name="client"
      partnerLinkType="tns:MasterProcess"
  
```

```

        myRole="MasterProcessProvider"
        partnerRole="MasterProcessRequester" />
    <partnerLink name="DetailProcess"
        partnerLinkType="dp:DetailProcess"
        myRole="DetailProcessRequester"
        partnerRole="DetailProcessProvider" />
    <partnerLink name="DetailProcess1"
        partnerLinkType="dp1:DetailProcess1"
        myRole="DetailProcess1Requester"
        partnerRole="DetailProcess1Provider" />
    <partnerLink name="DetailProcess2"
        partnerLinkType="dp2:DetailProcess2"
        myRole="DetailProcess2Requester"
        partnerRole="DetailProcess2Provider" />
</partnerLinks>

```

A signal activity shows the label value and the detail process coordinated with this master process. The label value (`startDetailProcess`) matches with the label value in the receive signal activity of all detail processes. This ensures that the signal is delivered to the correct process. There is one signal process per receive signal process. The master process signals all detail processes at runtime.

```
<bpelx:signal name="notifyDetailProcess" label="startDetailProcess" to="details" />
```

Assign, invoke, and receive activities describe the interaction between the master and detail processes. This example shows interaction between the master process and one of the detail processes (`DetailProcess`). Similar interaction is defined in this BPEL file for all detail processes.

Within the invoke activity, the `bpelx:invokeAsDetail` attribute is set to `true`. This attribute creates the partner process instance (`DetailProcess`) as a detail instance. You must manually add this attribute and set the value to `true` in the master process file for each detail process with which to interact. [Example 15–2](#) provides an example.

Example 15–2 *bpelx:invokeAsDetail* Attribute

```

<assign>
  <copy>
    <from variable="input" part="payload" query="/tns:processInfo/tns:value"/>
    <to variable="detail_input" part="payload" query="/dp:input/dp:number"/>
  </copy>
</assign>

<invoke name="receiveInput" partnerLink="DetailProcess"
  portType="dp:DetailProcess"
  operation="initiate"
  inputVariable="detail_input"
  bpelx:invokeAsDetail="true"/>

<!-- receive the result of the remote process -->
<receive name="receive_DetailProcess" partnerLink="DetailProcess"
  portType="dp:DetailProcessCallback"
  operation="onResult" variable="detail_output"/>

```

The master BPEL process includes a receive signal activity. This activity indicates that the master process waits until it receives a signal from all of its detail processes. The label value (`detailProcessComplete`) matches with the label value in the signal activity of each detail process. This ensures that the signal is delivered to the correct process. [Example 15–3](#) provides an example.

Example 15-3 Receive Signal Activity

```
<bpelx:receiveSignal name="waitForNotifyFromDetailProcess"
  label="detailProcessComplete"
  from="details"/>
```

15.1.1.1 Correlating a Master Process with Multiple Detail Processes

For environments in which you have one master and multiple detail processes, use the `bpelx:detailLabel` attribute for signal correlation. The following example shows how to use this attribute.

The first invoke activity invokes the `DetailsProcess` detail process and associates it with a label of `detailProcessComplete0`. [Example 15-4](#) provides an example.

Example 15-4 First Invoke Activity

```
<invoke name="invokeDetailProcess" partnerLink="DetailProcess"
  portType="dp:DetailProcess"
  operation="initiate"
  inputVariable="detail_input"
  bpelx:detailLabel="detailProcessComplete0"
  bpelx:invokeAsDetail="true"/>
```

The second invoke activity invokes the `DetailsProcess1` detail process and associates it with a label of `detailProcessComplete1`. [Example 15-5](#) provides an example.

Example 15-5 Second Invoke Activity

```
<invoke name="invokeDetailProcess1" partnerLink="DetailProcess1"
  portType="dp1:DetailProcess1"
  operation="initiate"
  inputVariable="detail_input1"
  bpelx:detailLabel="detailProcessComplete1-2"
  bpelx:invokeAsDetail="true"/>
```

The third invoke activity invokes the `DetailsProcess2` detail process again through a different port and with a different input variable. It associates the `DetailsProcess2` detail process with a label of `detailProcessComplete1-2`:

Example 15-6 Third Invoke Activity

```
<invoke name="invokeDetailProcess2" partnerLink="DetailProcess2"
  portType="dp2:DetailProcess2"
  operation="initiate"
  inputVariable="detail_input2"
  bpelx:detailLabel="detailProcessComplete1-2"
  bpelx:invokeAsDetail="true"/>
```

The receive signal activity of the master process shown in [Example 15-7](#) waits for a return signal from detail process `DetailProcess0`.

Example 15-7 Receive Signal Activity

```
<!-- This is a receiveSignal waiting for 1 child to signal back -->
<bpelx:receiveSignal name="waitForNotifyFromDetailProcess0"
  label="detailProcessComplete0" from="details"/>
```

The second receive signal activity of the master process shown in [Example 15-8](#) also waits for a return signal from `DetailProcess1` and `DetailProcess2`.

Example 15–8 Second Receive Signal Activity

```
<!-- This is a receiveSignal waiting for 2 child (detail) processes to signal back
-->
<bpelx:receiveSignal name="waitForNotifyFromDetailProcess1-2"
  label="detailProcessComplete1-2" from="details"/>
```

Note: If there is only one receive signal activity in the BPEL process, do not specify the `bpelx:detailLabel` attribute in the invoke activity. In these situations, a default `bpelx:detailLabel` attribute is assumed and does not need to be specified.

15.1.2 BPEL File Definition for Detail Processes

The BPEL process file of each detail process defines coordination with the master process.

A receive signal activity indicates that the detail process shown in [Example 15–9](#) waits until it receives a signal executed by its master process. The label value (`startDetailProcess`) matches with the label value in the signal activity of the master process.

Example 15–9 startDetailProcess Label Value

```
<bpelx:receiveSignal name="waitForNotifyFromMasterProcess"
  label="startDetailProcess" from="master"/>
```

A signal activity indicates that the detail process shown in [Example 15–10](#) signals its associated master process at runtime that processing is complete. The label value (`detailProcessComplete`) matches with the label value in the receive signal activity of each master process.

Example 15–10 Signal Activity

```
<bpelx:signal name="notifyMasterProcess" label="detailProcessComplete"
  to="master"/>
```

15.2 Defining Master and Detail Process Coordination in Oracle JDeveloper

This section provides an overview of how to define master and detail process coordination in Oracle BPEL Designer. In this example, one master process and one detail process are defined.

Note: This section only describes the tasks specific to master and detail process coordination. It does *not* describe the standard activities that you define in a BPEL process, such as creating variables, creating assign activities, and so on.

15.2.1 How to Create a Master Process**To create a master process:**

1. In the SOA Composite Editor, create a BPEL process service component. For this example, the process is named **MasterProcess**.

2. Double-click the **MasterProcess** BPEL process.
3. In the Component Palette, expand **BPEL Activities**.
4. Drag a **Signal** activity into the designer.
5. Double-click the **Signal** activity.

This activity signals the detail process to perform processing at runtime.

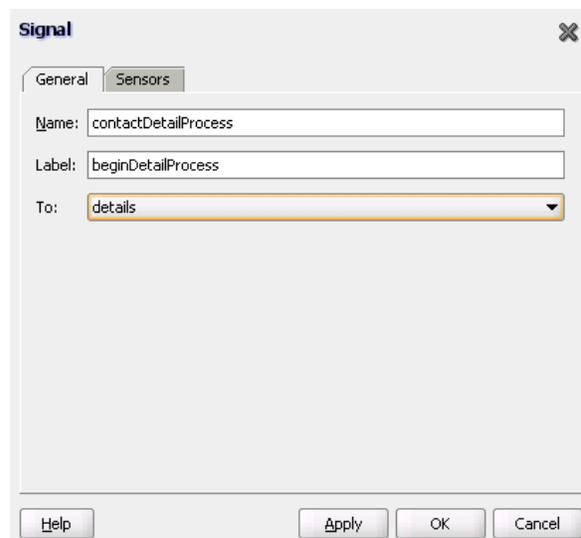
6. Enter the details described in [Table 15-2](#):

Table 15-2 Signal Dialog Fields and Values

Field	Value
Name	Enter a name (for this example, <code>contactDetailProcess</code>).
Label	Enter a label name (for this example, <code>beginDetailProcess</code>). This label must match the receive signal activity label you set in the detail process in Step 5 on page 15-8.
To	Select details as the type of process to receive this signal.

[Figure 15-3](#) shows the Signal dialog.

Figure 15-3 Signal Dialog



7. Click **OK**.
8. Drag a **Receive Signal** activity into the designer.
9. Double-click the **Receive Signal** activity.

This activity enables the master process to wait until it receives the signal executed by all of its detail processes.

10. Enter the details shown in [Table 15-3](#):

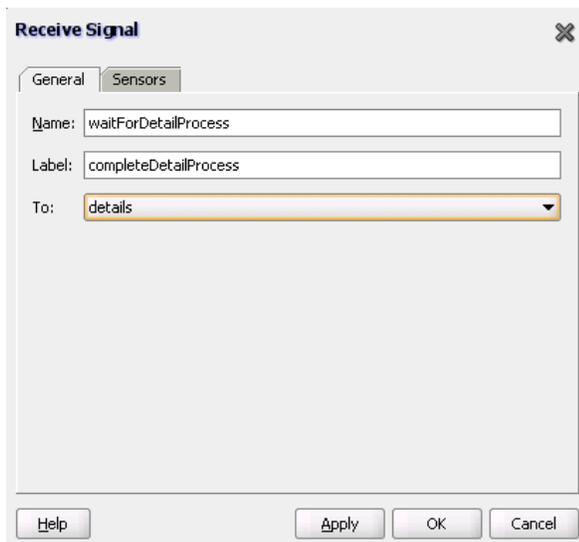
Table 15-3 Receive Signal Dialog Fields and Values

Field	Value
Name	Enter a name (for this example, <code>waitForDetailProcess</code>).

Table 15–3 (Cont.) Receive Signal Dialog Fields and Values

Field	Value
Label	Enter a label name (for this example, <code>completeDetailProcess</code>). This label must match the signal activity label you set in the detail process in Step 9 on page 15-9.
To	Select details as the type of process from which to receive the signal.

Figure 15–4 shows the Receive Signal dialog.

Figure 15–4 Receive Signal Dialog

11. Click **OK**.

The master process has now been designed to:

- Signal the detail process to perform processing at runtime.
- Wait until it receives the signal executed by the detail process.

15.2.2 How to Create a Detail Process

To create a detail process:

1. In the SOA Composite Editor, create a second BPEL process service component. For this example, the process is named **DetailProcess**.
2. Double-click the **DetailProcess** BPEL process.
3. Drag a **Receive Signal** activity into your BPEL process service component.
4. Double-click the **Receive Signal** activity.

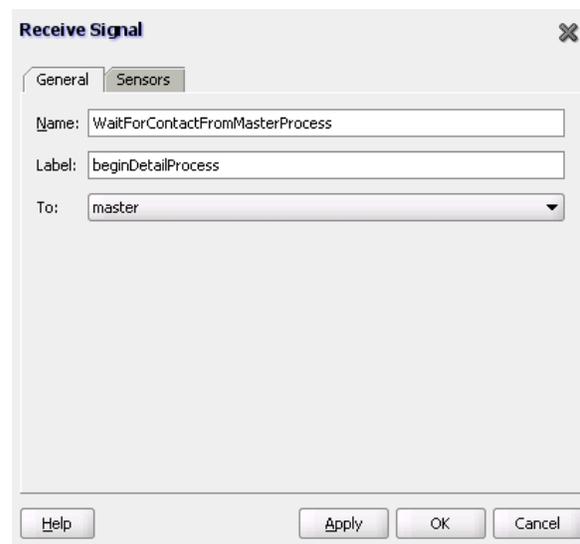
This activity enables the detail process to wait until it receives the signal executed by its master process.

5. Enter the details shown in [Table 15–4](#):

Table 15–4 Receive Signal Dialog Fields and Values

Field	Value
Name	Enter a name (for this example, <code>WaitForContactFromMasterProcess</code>).
Label	Enter a label name (for this example, <code>beginDetailProcess</code>). This label must match the signal activity label you set in the master process in Step 6 on page 15-7.
To	Select master as the type of process from which to receive the signal.

Figure 15–5 shows the Receive Signal dialog.

Figure 15–5 Receive Signal Dialog

6. Click **OK**.
7. Drag a **Signal** activity into the designer.
8. Double-click the **Signal** activity.

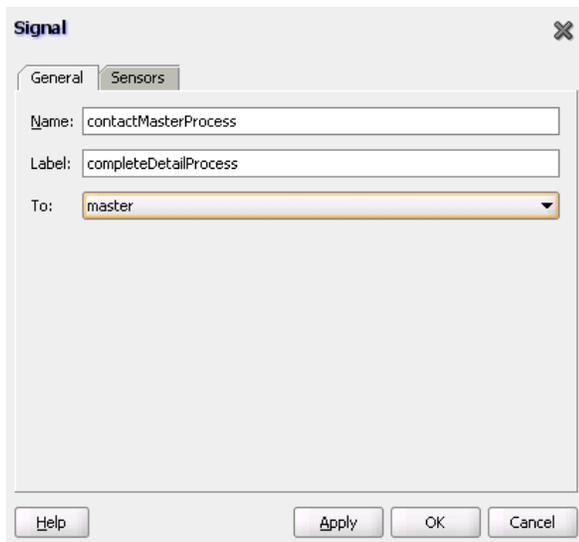
This activity enables the detail process to signal its associated master process at runtime that processing is complete.

9. Enter the details described in [Table 15–5](#):

Table 15–5 Signal Dialog Fields and Values

Field	Value
Name	Enter a name (for this example, <code>contactMasterProcess</code>).
Label	Enter a label name (for this example, <code>completeDetailProcess</code>). This label must match the receive signal activity label you set in the master process in Step 10 on page 15-7.
To	Select master as the destination.

Figure 15–6 shows the Signal dialog.

Figure 15–6 Signal Dialog

10. Click **OK**.

The detail process has now been designed to:

- Wait until it receives the signal executed by its master process.
- Signal the master process at runtime that processing is complete.

15.2.3 How to Create an Invoke Activity

To create an invoke activity:

1. Return to the **MasterProcess** master process.
2. Drag an **Invoke** activity into your BPEL process service component.
3. Double-click the **Invoke** activity.
4. Select the **DetailProcess** BPEL process you created in Step 1 on page 15-8 as the partner link.
5. Complete all remaining fields in the Invoke dialog, and click **OK**.
6. In the designer, click **Source**.
7. Add `bpelx:invokeAsdetail` to the invoke activity and set it to `true`, as shown in [Example 15–11](#).

Example 15–11 `bpelx:invokeAsdetail` Attribute

```
<invoke name="MyInvoke" partnerLink="DetailProcess"
  portType="dp:DetailProcess"
  operation="initiate"
  inputVariable="detail_input"
  bpelx:invokeAsDetail name="true"/>
```

This attribute creates the partner process (`DetailProcess`) as a detail instance.

8. If this is an environment in which one master process is interacting with multiple detail processes, perform the following tasks:

- a. Specify the `bpelx:detailLabel` attribute for correlating with the receive signal activity, as shown in [Example 15–12](#).

Example 15–12 `bpelx:detailLabel` Attribute

```
<invoke name="MyInvoke" partnerLink="DetailProcess"
  portType="dp:DetailProcess"
  operation="initiate"
  inputVariable="detail_input"/>
bpelx:detailLabel="detailProcessComplete0"
<bpelx:invokeAsdetail name="true"/>
```

- b. Specify the same label value of `detailProcessComplete0` in the receive signal activity of the master process, as shown in [Example 15–13](#).

Example 15–13 `detailProcessComplete0` Label Value

```
<bpelx:receiveSignal name="waitForNotifyFromDetailProcess0-1"
label="detailProcessComplete0" from="details"/>
```

- c. Repeat these steps as necessary for additional detail processes, ensuring that you specify a different label value.
9. From the **File** main menu, select **Save All**.
Master and detail coordination design is now complete.

Using the Notification Service

This chapter describes how to send notifications from a BPEL process using a variety of channels. A BPEL process can be designed to send email, voice message, instant messaging (IM), or short message service (SMS) notifications. A BPEL process can also be designed to consider an end user's channel preference at runtime for selecting the notification channel.

This chapter includes the following sections:

- [Section 16.1, "Introduction to the Notification Service"](#)
- [Section 16.2, "Introduction to Notification Channel Setup"](#)
- [Section 16.3, "Selecting Notification Channels During BPEL Process Design"](#)
- [Section 16.4, "Allowing the End User to Select Notification Channels"](#)

Note: The fax and pager notification channels are not supported in 11g Release 1 (11.1.1).

16.1 Introduction to the Notification Service

Various scenarios may require sending email messages or other types of notifications to users as part of the process flow. For example, certain types of exceptions that cannot be handled automatically may require manual intervention. In this case, a BPEL process can use the notification service to alert users by voice, IM, SMS, or email.

The contact information (email address, phone number, and so on) of the recipient is either static (such as `admin@yourcompany.com`) or obtained dynamically during runtime. To obtain the contact information dynamically, XPath expressions can be used to retrieve it from the identity store (LDAP) or extract it from the BPEL payload.

This chapter uses the following terms:

- Notification
An asynchronous message sent to a user by a specific channel. The message can be sent as an email, voice, IM, or SMS message.
- Actionable notification
A notification to which the user can respond. For example, workflow sends an email to a manager to approve or reject a purchase order. The manager approves or rejects the request by replying to the email with appropriate content.
- Human task email notification layer

Sends email notifications directly from a BPEL process or implicitly from the human task part of a BPEL process. Implicit notifications are modeled from the Human Task Editor.

For sending email notifications directly from a BPEL process, you must explicitly specify the user information in the BPEL process and can be inside or outside of a human task scope.

For sending email notifications implicitly from the human task part of a BPEL process, you only specify the recipient based on the relationship of the user with regards to the task (that is, the creator, assignee, and so on).

Note: Implicit notifications are processed through more layers of code than explicit notifications. If explicit notifications are functioning correctly, it does not mean that implicit notifications also function correctly.

- Oracle User Messaging Service

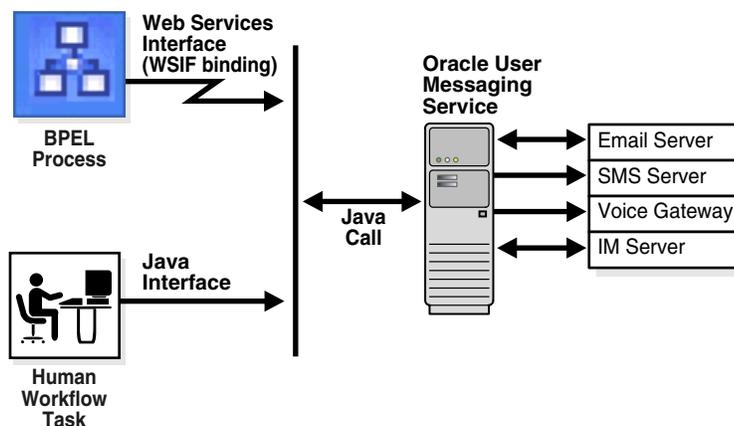
Oracle User Messaging Service is a new feature for release 11g. The BPEL notification service uses the underlying infrastructure provided by Oracle User Messaging Service to send notifications.

Oracle User Messaging Service also provides the user preference infrastructure for getting the end user's preferred channel during runtime.

For more information on the Oracle User Messaging Service, see [Appendix 39, "Oracle User Messaging Service."](#)

[Figure 16–1](#) shows the Oracle User Messaging Service interfaces and supported service types.

Figure 16–1 Service Interfaces and Supported Service Types



For more information about notifications, see the following section:

- [Section 29.2, "Notifications from Human Workflow"](#)
- [Section 25.3.9, "How to Specify Participant Notification Preferences"](#) for instructions on specifying email notifications in the Human Task Editor
- [Part VII, "Using Oracle User Messaging Service"](#)

16.2 Introduction to Notification Channel Setup

Notification setup is a multiple-step process that involves three user interface tools. [Table 16–1](#) provides an overview of this process, including the task to perform, the tool to use, and the documentation to which to refer for more specific details.

Table 16–1 Notification Tasks

Task	Description	User Interface	Described In...
Select a channel for sending notifications in a SOA composite application.	Select a method for sending notifications: <ul style="list-style-type: none"> ▪ Explicitly select and configure an email, IM, SMS, or voice channel. or ▪ Do not explicitly select a notification channel, but simply select that a notification must be sent. Channel selection occurs later in the User Messaging Preferences user interface. 	Selected and configured by the BPEL process designer in Oracle BPEL Designer	Section 16.3, "Selecting Notification Channels During BPEL Process Design" or Section 16.4, "Allowing the End User to Select Notification Channels"
Configure the driver for the notification channel	You configure drivers on the same Oracle WebLogic Server on which you deploy the SOA composite application. This action enables participants to receive and forward notifications. Driver support is provided for email, IM, SMS, and voice channels.	Configured by the administrator in Oracle Enterprise Manager Fusion Middleware Control Console	<i>Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite</i>
Configure the notification mode and actionable accounts for human workflows	If you are using notifications with human workflow, you configure the notification mode and actionable account for email.	Configured by the administrator in Oracle Enterprise Manager Fusion Middleware Control Console	<i>Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite</i>
Register the devices used to access messages by specifying user preferences	This action enables workflow participants to receive notification messages. For example, the end user registers email clients and specifies the message content to receive and the channel to use for receiving messages. If no channel is specified, email is used by default. Note that the preferences set in this application are applicable only to that specific end user, and not to other users.	Registered by the end user in the User Messaging Preferences user interface	Part VII, "Using Oracle User Messaging Service"

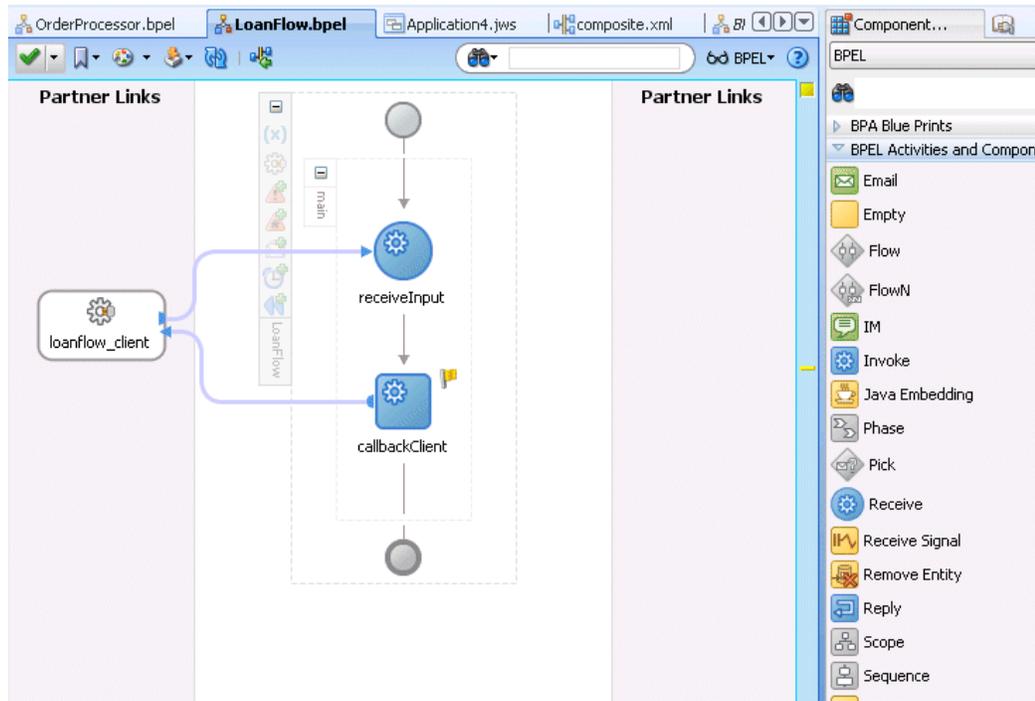
16.3 Selecting Notification Channels During BPEL Process Design

Oracle JDeveloper includes the notification channels in the Component Palette, as shown in [Figure 16–2](#). You can set the exact notification channels to use during design time. For example, a BPEL process can be designed to use the following notification channels:

- If an expense report amount is less than \$1000, an email notification channel is used.

- If an expense report amount is between \$1000 and \$2000, an SMS notification channel is used.
- If an expense report amount is more than \$2000, a voice notification channel is used.

Figure 16–2 Oracle JDeveloper—Notification Channels



To select the notification channel during BPEL process design:

1. From the Component Palette list, select **BPEL**.
2. Expand **BPEL Activities and Components**.
3. From the Component Palette, drag a notification channel into the designer:
 - **Email**
 - **IM**
 - **SMS**
 - **Voice**
4. See the section in [Table 16–2](#) based on the notification channel you selected.

Table 16–2 Notification Channels

If You Selected...	See...
Email	Section 16.3.1, "How To Configure the Email Notification Channel" to configure email notification
IM	Section 16.3.2, "How to Configure the IM Notification Channel" to configure IM notification
SMS	Section 16.3.3, "How to Configure the SMS Notification Channel" to configure SMS notification

Table 16–2 (Cont.) Notification Channels

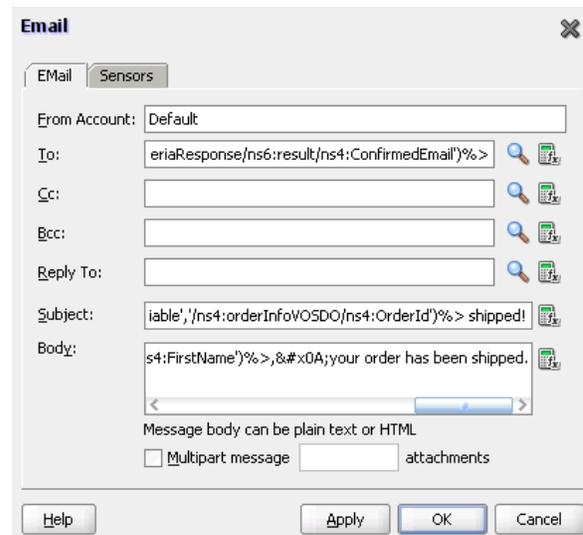
If You Selected...	See...
Voice	Section 16.3.4, "How to Configure the Voice Notification Channel" to configure voice message notification

Note: If you delete an email, voice, SMS, or IM activity, any partner link with which it is integrated is not automatically deleted.

16.3.1 How To Configure the Email Notification Channel

When you select **Email** from the Component Palette, the Email dialog appears. [Figure 16–3](#) shows the required email notification parameters.

Figure 16–3 Email Dialog



To configure the email notification channel:

1. Enter information for each field as described in [Table 16–3](#).

Note: For the **To**, **CC**, and **Bcc** fields, separate multiple addresses with a semicolon (;).

Table 16–3 Email Notification Parameters

Name	Description
From Account	The name of the account used to send this message. The default account is named Default and is editable from the Workflow Notification Properties page in Oracle Enterprise Manager Fusion Middleware Control Console. To add additional accounts, you must use the System MBean Browser in Oracle Enterprise Manager Fusion Middleware Control Console. For information on editing this property in Oracle Enterprise Manager Fusion Middleware Control Console, see <i>Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite</i> .

Table 16–3 (Cont.) Email Notification Parameters

Name	Description
To	<p>The email address to which the message is to be delivered. This can be one of the following:</p> <ul style="list-style-type: none"> ■ A static email address entered at the time the message is created ■ An email address retrieved using the identity service ■ A dynamic address from the payload <p>The XPath Expression Builder can be used to get the dynamic email address from the input. See Section 16.3.5, "How to Select Email Addresses and Telephone Numbers Dynamically."</p>
CC and Bcc	<p>The email addresses to which the message is copied and blind copied. This can be a static or dynamic address, as described for the To address.</p>
Reply To	<p>The email address to use for replies. This can be a static or dynamic address, as described for the To address.</p>
Subject	<p>The subject of the email message. This can be plain text or dynamic text. The XPath Expression Builder can be used to set dynamic text based on data from process variables that you specify.</p>
Body	<p>The message body of the email message. This can be plain text, HTML, or dynamic text, as described for the Subject parameter.</p>
Multipart message with <i>n</i> attachments	<p>Select to specify email attachments. See Section 16.3.1.1, "Setting Email Attachments."</p> <p>The number of attachments if Multipart message is selected. The number does not include the body. For example, if you have a body and one attachment, specify 1.</p>

2. Click **OK**.

The BPEL fragment that invokes the notification service to send the email message is created.

3. See [Table 16–1](#) on page 16-3 for additional configuration procedures to perform.

The WebLogic Fusion Order Demo application uses an email activity in the **Scope_NotifyCustomerofCompletion** scope. The Oracle User Messaging Service sends the email to a customer when an order is fulfilled. The following details are specified in the Email dialog:

- An XPath expression specifies the customer’s email address.

```
bpws:getVariableData('gCustomerInfoVariable', 'parameters', '/ns3:findCustomerInfoV01CustomerInfoV0CriteriaResponse/ns3:result/ns2:ConfirmedEmail')
```

- A combination of manually-entered text and an XPath expression specifies the ID of the order:

```
Order with id
<%bpws:getVariableData('gOrderInfoVariable', '/ns2:orderInfoVOSDO/ns2:OrderID')%> shipped!
```

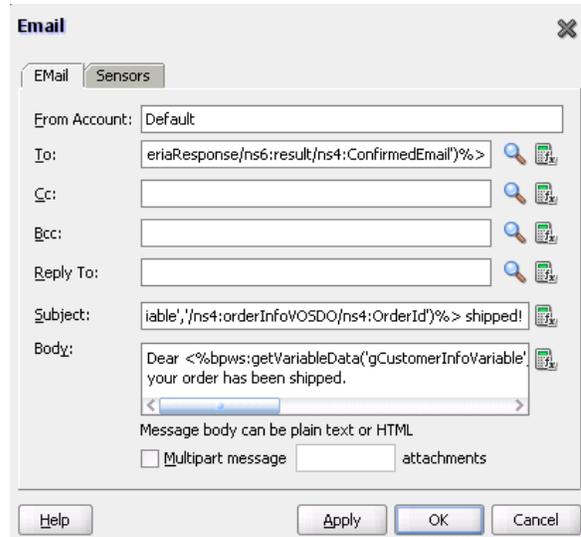
- A combination of manually-entered text and an XPath expression specifies the body of the email message:

```
Dear
<%bpws:getVariableData('gCustomerInfoVariable', 'parameters', '/ns6:findCusto
```

```
merInfoV01CustomerInfoV0CriteriaResponse/ns6:result/ns4:FirstName')%>,
your order has been shipped.
```

Figure 16–4 provides details.

Figure 16–4 Email Dialog



16.3.1.1 Setting Email Attachments

When you send email attachments, you mark the email as a multipart message and set the number of attachments to send. The number of attachments does not need to include the body plus the attachments. For example, to send an email message with one file as an attachment, set the number to 1. When sending attachments, set the content body to have a `MultiPart` element that contains as many `BodyPart` elements as the number of attachments. Each `BodyPart` has three elements: `ContentBody`, `MimeType`, and `BodyPartName`. All three elements must be set for each attachment.

To add an attachment to an email message:

1. From the Component Palette, select **Email** as the notification channel.
2. Specify values for **To**, **Subject**, and **Body**.
3. Select **Multipart message** and enter 1 for the number of attachments. (Note that the number of attachments does not need to include the body part.)

The BPEL fragment with an `assign` activity with multiple `copy` rules is generated. One of the `copy` rules copies the attachment.

4. Click **OK**.
5. Expand the email activity.

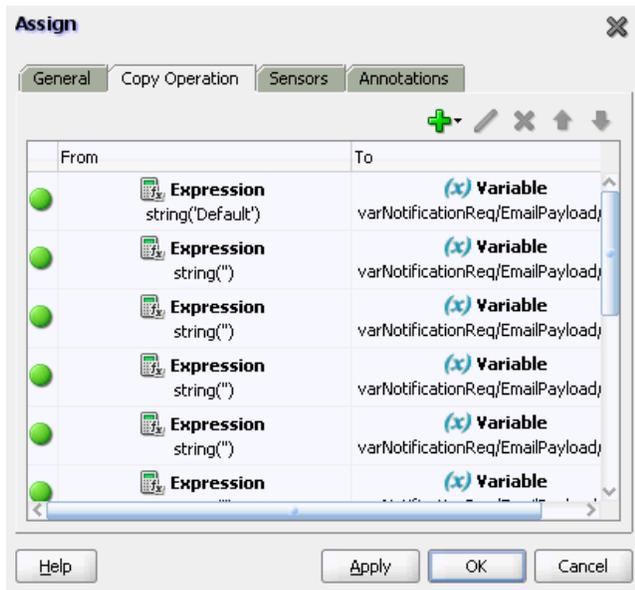
Note that an `assign` activity named **EmailParamsAssign** appears.

Each body part has three attributes: **ContentBody**, **MimeType**, and **BodyPartName**. Default names, MIME types, and contents are generated for each attachment in this `assign` activity.

6. Double-click **EmailParamsAssign**.

Note the default settings in **EmailParamsAssign**.

Figure 16–5 EmailParamsAssign Assign Activity



7. Change the default values for **ContentBody**, **MimeType**, and **BodyPartName** to values specific to your environment.
8. Save your changes.

For more information about sending attachments using email, see the following documentation:

- [Appendix I, "Oracle User Messaging Service Applications"](#)
- The notification-101 sample, which is available at the following URL:
http://www.oracle.com/technology/sample_code/products/hwf

16.3.1.2 Formatting the Body of an Email Message as HTML

You can format the body of an email message as HTML rather than as straight text. To perform this action, apply an XSLT transform to generate the email body. Add in the XSLT tag you want to use. Tools such as XMLSpy can provide assistance in writing and testing the XSLT. The MIME type should be `string('text/html; charset=UTF-8')`.

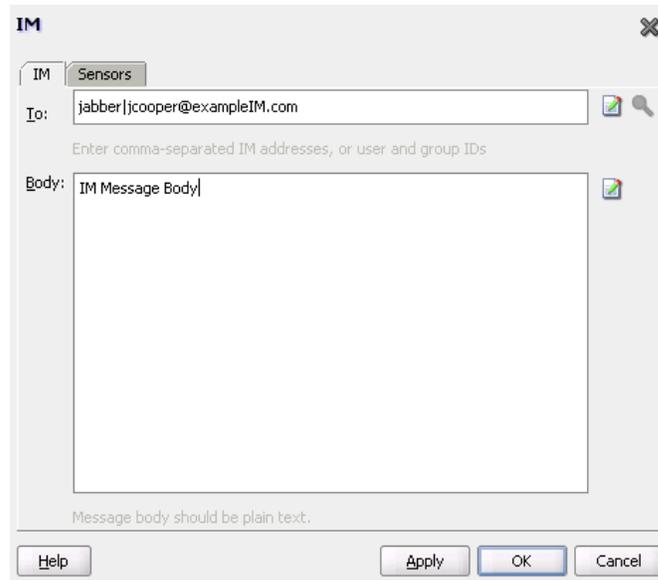
The email notification assignment looks as shown in [Example 16–1](#):

Example 16–1 Email Notification Assignment

```
<copy>
  <from
    expression="ora:processXSLT('TransformPositionSummary7.xslt',bpws:
    getVariableData('ClientPositionSummary'))"/>
    <to variable="varNotificationReq" part="EmailPayload"
    query="/EmailPayload/ns9:Content/ns9:ContentBody"/>
  </copy>
```

16.3.2 How to Configure the IM Notification Channel

When you drag **IM** from the Component Palette, the IM dialog appears. [Figure 16–6](#) shows the required IM notification parameters.

Figure 16–6 IM Dialog**To configure the IM notification channel:**

1. Enter information for each field as described in [Table 16–4](#).

Table 16–4 IM Notification Parameters

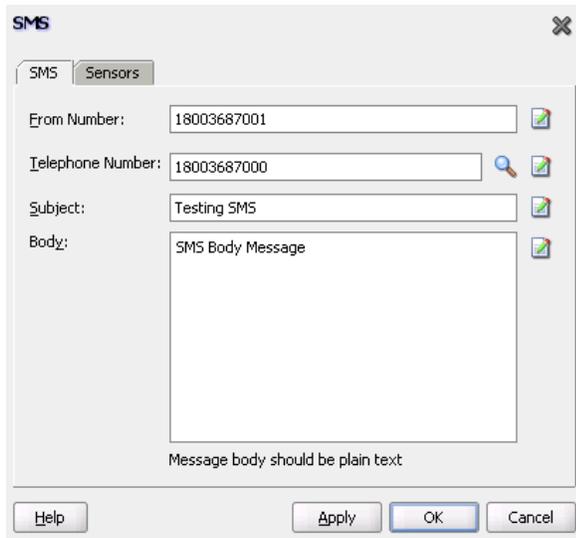
Name	Description
To	The IM address to which the message is to be delivered. Enter the address manually or click the XPath Expression Builder icon to display the Expression Builder dialog to dynamically enter an account.
Body	The IM message body. This can be plain text or dynamic text. The XPath Expression Builder can be used to set dynamic text based on data from process variables that you specify.

2. Click **OK**.
The BPEL fragment that invokes the notification service for IM notification is created.
3. See [Table 16–1](#) on page 16-3 for additional configuration procedures to perform.

16.3.3 How to Configure the SMS Notification Channel

When you select **SMS** from the Component Palette, the SMS dialog appears. [Figure 16–7](#) shows the required SMS notification parameters.

Figure 16–7 SMS Dialog



To configure the SMS notification channel:

1. Enter information for each field as described in [Table 16–5](#).

Table 16–5 SMS Notification Parameters

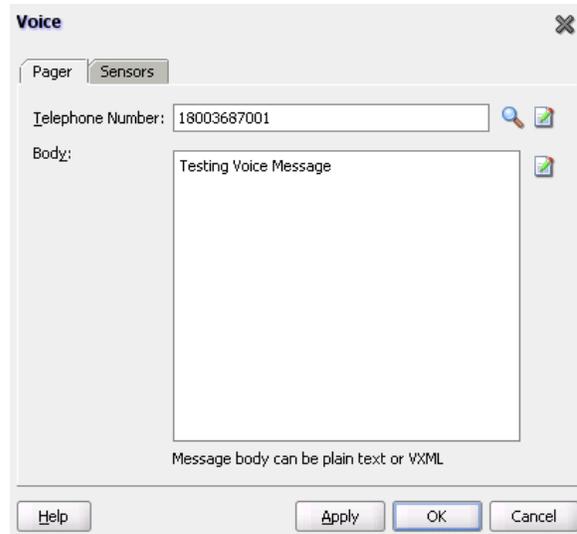
Name	Description
From Number	The telephone number from which to send the SMS notification. This can be a static telephone number entered at the time the message is created or a dynamic telephone number from the payload. The XPath Expression Builder can be used to get the dynamic telephone number from the input. See Section 16.3.5, "How to Select Email Addresses and Telephone Numbers Dynamically."
Telephone Number	Select a method for specifying the telephone number to which to deliver the message: <ul style="list-style-type: none"> ■ A static telephone number entered at the time the message is created. ■ A telephone number retrieved using the identity service. ■ A dynamic telephone number from the payload. The XPath Expression Builder can be used to get the dynamic telephone number from the input.
Subject	The subject of the SMS message. This can be plain text or dynamic text. The XPath Expression Builder can be used to set dynamic text based on data from process variables that you specify.
Body	The SMS message body. This must be plain text. This can be plain text or dynamic text as described for the Subject parameter.

2. Click **OK**.
The BPEL fragment that invokes the notification service for SMS notification is created.
3. See [Table 16–1](#) on page 16-3 for additional configuration procedures to perform.

16.3.4 How to Configure the Voice Notification Channel

When you select **Voice** from the Component Palette, the Voice dialog appears. [Figure 16–8](#) shows the required voice notification parameters.

Figure 16–8 Voice Dialog



To configure the voice notification channel:

1. Enter information for each field as described in [Table 16–6](#).

Table 16–6 Voice Notification Parameters

Name	Description
Telephone Number	<p>The telephone number to which the message is to be delivered. Specify the number through one of the following methods:</p> <ul style="list-style-type: none"> ■ A static telephone number entered at the time the message is created ■ A telephone number retrieved using the identity service ■ A dynamic telephone number from the payload <p>The XPath Expression Builder can be used to retrieve the dynamic telephone number from the input.</p>
Body	<p>The message body. This can be plain text, XML, or dynamic text. The XPath Expression Builder can be used to set dynamic text based on data from process variables that you specify.</p>

2. Click **OK**.

The BPEL fragment that invokes the notification service for voice notification is created.

3. See [Table 16–1](#) on page 16-3 for additional configuration procedures to perform.

16.3.5 How to Select Email Addresses and Telephone Numbers Dynamically

You may need to set email addresses or telephone numbers 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 get the email address or telephone number directly from the payload, use the following XPath expression:

```
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 underlying identity store (LDAP) use the following XPath expression:

```
ids:getUserProperty(userName, attributeName[, realmName])
```

The first argument evaluates to the user ID. The second argument is the property name. The third argument is the realm name. [Table 16–7](#) lists the property names that can be used in this XPath function.

Table 16–7 Properties for the Dynamic User XPath Function

Property Name	Description
mail	Look up a user’s email address.
telephoneNumber	Look up a user’s telephone number.
mobile	Look up a user’s mobile telephone number.
homephone	Look up a user’s home telephone number.

The following example gets the email address of the user identified by the variable `inputVariable`, part payload, and queries

```
/client:BPELProcessRequest/client:userID:
```

```
ids:getUserProperty(bpws:getVariableData('inputVariable', 'payload', '/client:BPELProcessRequest/client:userid'), 'mail')
```

If `realmName` is not specified, then the default realm name is used. For example, if the default realm name is `jazn.com`, the following XPath expression searches for the user in the `jazn.com` realm:

```
ids:getUserProperty('jcooper', 'mail');
```

The following XPath expression provides the same functionality as the one above. In this case, however, the realm name of `jazn.com` is explicitly specified:

```
ids:getUserProperty('jcooper', 'mail', 'jazn.com');
```

16.3.6 How to Select Notification Recipients by Browsing the User Directory

You can select users or groups in Oracle JDeveloper to whom you want to send notifications by browsing the user directory (for example, Oracle Internet Directory) that is configured for use with Oracle BPEL Process Manager. Click the **Search** icon to the right of the following fields to open the Identity Lookup dialog:

- **To** field on the Email and IM dialogs

- **Telephone Number** field on the SMS and Voice dialogs

For more information about using the Identity Lookup dialog, see [Chapter 29, "Introduction to Human Workflow Services"](#)

16.4 Allowing the End User to Select Notification Channels

You can design a BPEL process in which you do not explicitly select a notification channel during design time, but simply indicate that a notification must be sent. The channel to use for sending notifications is resolved at runtime based on preferences defined by the end user in the User Messaging Preferences user interface of the Oracle User Messaging Service. This moves the responsibility of notification channel selection from the BPEL process designer in Oracle BPEL Designer to the end user. If the end user does not select a preferred channel or rule, email is used by default for sending notifications to that user. Regardless of who selects the channel to use, channel use is still based on the driver installation and configuration performed in the Oracle User Messaging Service section of Oracle Enterprise Manager Fusion Middleware Control Console by the administrator.

For example, an end user may set their preferences as follows:

- If an expense report amount is less than \$153, they receive an email notification.
- If an expense report amount is between \$153 and \$3678, they receive an SMS notification.
- If an expense report amount is more than \$3678, they receive a voice notification.

Note: You can also set user preferences for sending notifications in human workflows in the Human Task Editor. Set these preferences in the **Notification Filters** part of the **Notification Settings** section. These preferences are used to evaluate rules in the task. For more information, see [Section 25.3.9.7, "Sending Task Attachments with Email Notifications."](#)

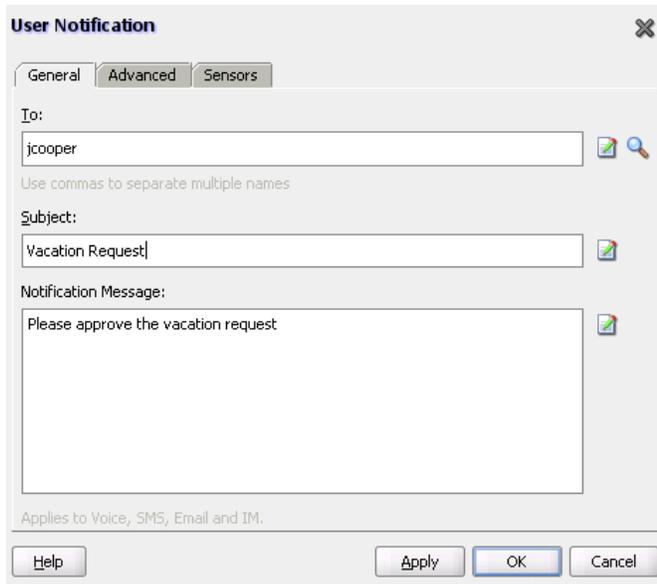
For more information about the User Messaging Preferences user interface, see [Chapter 42, "User Messaging Preferences."](#)

16.4.1 How to Allow the End User to Select Notification Channels

To allow the end user to select notification channels:

1. From the Component Palette list, select **BPEL**.
2. Expand **BPEL Activities and Components**.
3. From the Component Palette, drag the **User Notification** activity into the designer. [Figure 16–9](#) shows the required user notification parameters.

Figure 16–9 User Notification Dialog



4. Enter information for each field as described in [Table 16–8](#).

Table 16–8 User Notification Parameters

Name	Description
To	<p>Enter a valid user for the recipient of this notification message through one of the following methods:</p> <ul style="list-style-type: none"> ■ Enter the user manually ■ Click the Search icon to display a dialog for selecting a user configured through the identity service. The identity service enables the lookup of user properties, roles, and group memberships. ■ Click the XPath Expression Builder icon to display the Expression Builder dialog to dynamically enter a user. <p>Note: You must specify a user name (for example, <code>jcooper</code>) instead of an address.</p>
Subject	<p>Enter a message name or click the XPath Expression Builder icon to display the Expression Builder dialog to dynamically enter a subject. If notification is sent through email, this field is used during runtime. This field is ignored if notifications are sent through the voice, SMS, or IM channels.</p>
Notification Message	<p>Enter the notification message or click the XPath Expression Builder icon to display the Expression Builder dialog to dynamically enter a message to send.</p>

5. Click **Apply**.

16.4.1.1 How to Create and Send Headers for Notifications

The **Advanced** tab of the User Notification dialog enables you to create and send header and name information that may be useful to an end user in creating their own preference rules for receiving notifications. For example:

- The BPEL designer creates specifies the users named `jcooper` and `jstein` in the **General** tab.

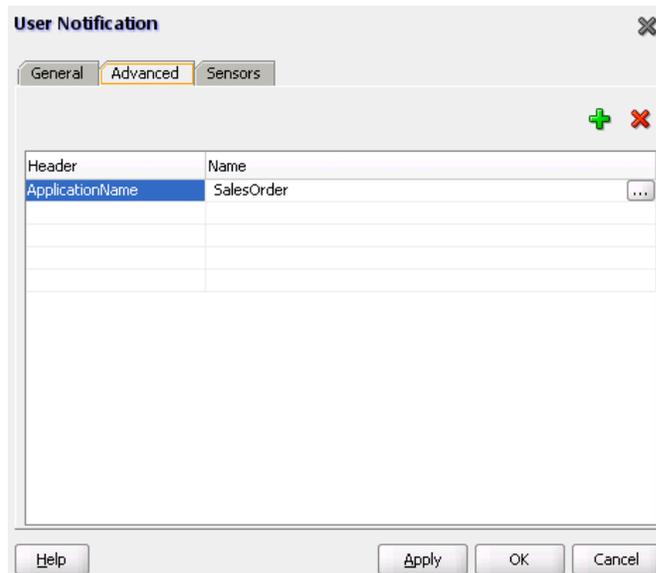
- The BPEL designer creates the following header and name information in the **Advanced** tab:
 - Amount = payload->salary
 - Application = HR-Application
 - The administrator deploys the process and configures various channel drivers in Oracle Enterprise Manager Fusion Middleware Control Console.
 - The end user jcooper creates the following preference rules in the User Messaging Preferences user interface:


```
'Email if Amount < 30000" and "SMS if Amount is between 30000 and 100000' and "Voice if Amount > 100000"
```
 - The end user jstein creates the following preference rule in the User Messaging Preferences user interface:


```
If "Application == HR-Application" and Amount > 2000000" send Voice
```
1. If you want to create and send header and name information to an end user for creating their own preference rules, click **Advanced**.

Figure 16–10 shows the **Advanced** tab of the User Notification dialog.

Figure 16–10 User Notification Advanced Parameters



2. Click the **Add** icon to add a row to the **Header** and **Name** columns.
3. In the **Header** column, click the field to display a list for selecting a value. Otherwise, manually enter a value.
4. In the **Name** column, enter a value.
5. Click **OK**.

Using Oracle BPEL Process Manager Sensors

This chapter describes how to use sensors to select BPEL activities, variables, and faults to monitor during runtime. This chapter describes how to use and set up sensors for a BPEL process.

This chapter includes the following sections:

- [Section 17.1, "Introduction to Sensors"](#)
- [Section 17.2, "Configuring Sensors and Sensor Actions in Oracle JDeveloper"](#)
- [Section 17.3, "Viewing Sensors and Sensor Action Definitions in Oracle Enterprise Manager Fusion Middleware Control Console"](#)

For more information about sensors, see the following sections:

- [Appendix D, "Understanding Sensor Public Views and the Sensor Actions XSD"](#)
- [Section 32.6, "Integrating BPEL Sensors with Oracle BAM"](#) for how to create sensor actions in Oracle BPEL Process Manager to publish sensor data as data objects in an Oracle BAM Server

17.1 Introduction to Sensors

Sensors are used to declare interest in specific events throughout the life cycle of a BPEL process instance. In a business process, that can be the activation and completion of a specific activity or the modification of a variable value in the business process.

When a sensor is triggered, a specific sensor value is created. For example, if a sensor declares interest in the completion of a BPEL scope, the sensor value consists of the name of the BPEL scope and a time stamp value of when the activity was completed. If a sensor value declares interest in a BPEL process variable, then the sensor value consists of the value of the variable at the moment it was modified, a time stamp when the variable was modified, and the activity name and type that modified the BPEL variable.

The data format for sensor values is normalized and well-defined using XML schema.

A sensor action is an instruction on how to process sensor values. When a sensor is triggered by Oracle BPEL Process Manager, a new sensor value for that sensor is created. After that, all the sensor actions associated with that sensor are performed. A sensor action typically persists the sensor value in a database or sends the normalized sensor value data to a JMS queue or topic. For integration with Oracle Business Activity Monitoring, the sensor value can be sent to the BAM adapter.

You can define the following types of sensors, either through Oracle JDeveloper or manually by providing sensor configuration files.

- Activity sensors

Activity sensors are used to monitor the execution of activities within a BPEL process. For example, they 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 also monitor variables of the activity.

- Variable sensors

Variable sensors are 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

Fault sensors are 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, two new files are created as part of the BPEL process archive:

- `bpel_process_name_sensor.xml`

Contains the sensor definitions of a BPEL process

- `bpel_process_name_sensorAction.xml`

Contains the sensor action definitions of a BPEL process

See [Section 17.2.1, "How to Configure Sensors"](#) and [Section 17.2.2, "How to Configure Sensor Actions"](#) for how these files are created.

After you define sensors for a BPEL process, you must configure sensor actions to publish the sensor data to an endpoint. You can publish sensor data to the BPEL dehydration store schema, 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 in which 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.

- JMS queue

Publishes the sensor data to a JMS queue. The XML data is posted in accordance with the `Sensor.xsd` file. This file is included with Oracle JDeveloper in the `JDEV_HOME\jdeveloper\integration\seed\soa\shared\bpel` directory.

- JMS topic

Publishes the sensor data to a JMS topic. The XML data is posted in accordance with the same `Sensor.xsd` file used with JMS queues.

- Custom

Publishes the data to a custom Java class.

- JMS Adapter

Uses the JMS adapter to publish 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

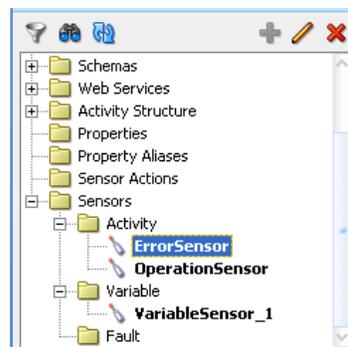
The sensors for a sensor action.

Oracle BAM sensors publish information and events from Oracle BPEL Process Manager to Oracle BAM. Oracle BAM can display the data in rich real-time dashboards for end-to-end monitoring of an application. For more information, see [Section 32.6, "Integrating BPEL Sensors with Oracle BAM."](#)

17.2 Configuring Sensors and Sensor Actions in Oracle JDeveloper

In Oracle JDeveloper, sensor actions and sensors are displayed as part of the process tree structure, as shown in [Figure 17-1](#).

Figure 17-1 Sensors and Sensor Actions Displayed in Oracle JDeveloper



You typically add or edit sensors as part of the BPEL modeling of activities, faults, and variables. You can add sensor actions by right-clicking the **Sensor Actions** folders and selecting **Create > Sensor Action**. To add activity sensors, variable sensors, or fault sensors, expand the **Sensors** folder, right-click the appropriate **Activity**, **Variable**, or **Fault** subfolder, and click **Create**.

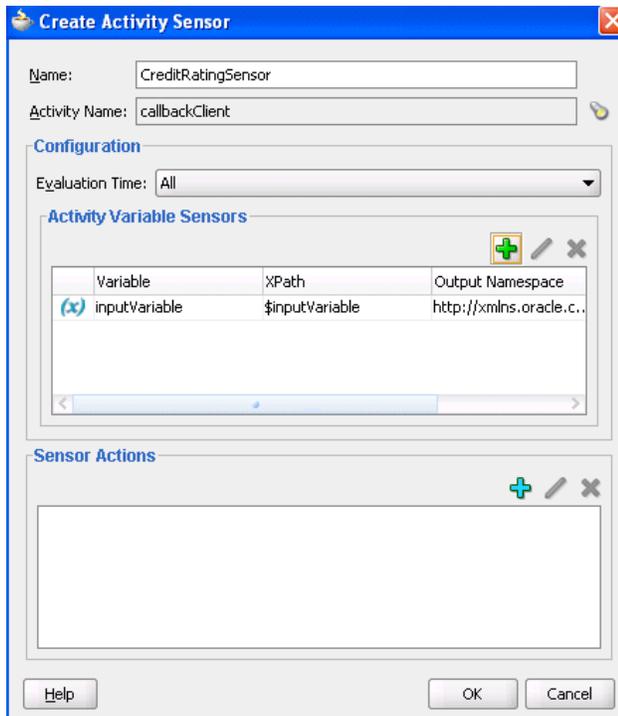
The following sections describe how to configure sensors and sensor actions.

17.2.1 How to Configure Sensors

Assume you are monitoring a **LoanFlow** application, and want to know the following:

- When a scope named **getCreditRating** is initiated
- When it is completed
- At completion, what is the credit rating for the customer

The solution is to create an activity sensor for the **getCreditRating** scope in Oracle BPEL Designer, as shown in [Figure 17-2](#). Activities that have sensors associated with them are identified with a magnifying glass in Oracle BPEL Designer.

Figure 17–2 Creating an Activity Sensor

The **Evaluation Time** list shown in [Figure 17–2](#) controls the point at which the sensor is fired. You can select from the following:

- **All:**
The sensor monitors during the activation, completion, fault, compensation, and retry phases.
- **Activation**
The sensor is fired just before the activity is executed.
- **Completion**
The sensor is fired just after the activity is executed.
- **Fault**
The sensor is fired if a fault occurs during the execution of the activity. Select this value only for sensors that monitor simple activities.
- **Compensation**
The sensor is fired when the associated scope activity is compensated. Select this value only for sensors that monitor scopes.
- **Retry**
The sensor is fired when the associated invoke activity is retried.

A new entry is created in the `bpel_process_name_sensor.xml` file, as shown in [Example 17–1](#):

Example 17–1 *bpel_process_name_sensor.xml* file

```
<sensor sensorName="CreditRatingSensor">
```

```

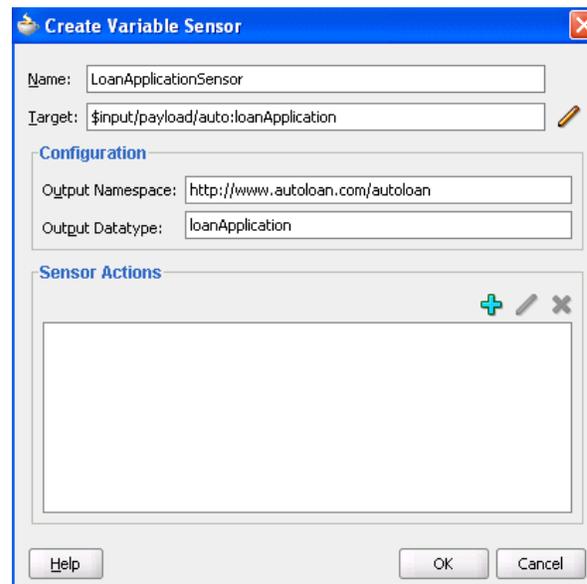
classname="oracle.tip.pc.services.reports.dca.agents.BpelActivitySensorAgent"
    kind="activity"
    target="callbackClient">

    <activityConfig evalTime="all">
        <variable outputNamespace="http://www.w3.org/2001/XMLSchema"
            outputDataType="int"
            target="$crOutput/payload//services:rating"/>
    </activityConfig>
</sensor>

```

If you want to record all the incoming loan requests, create a variable sensor for the variable `input`, as shown in [Figure 17-3](#).

Figure 17-3 *Creating a Variable Sensor*



A new entry is created in the `bpel_process_name_sensor.xml` file, as shown in [Example 17-2](#):

Example 17-2 *bpel_process_name_sensor.xml file*

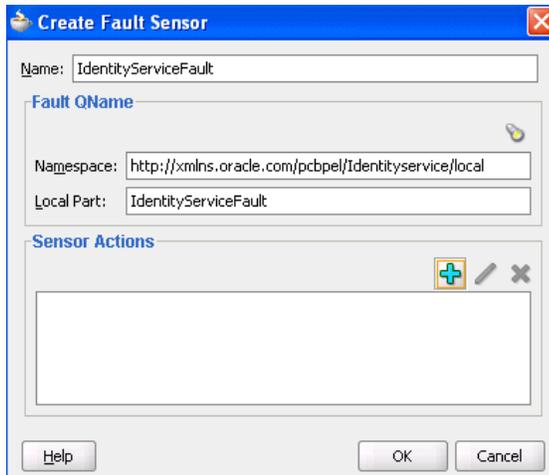
```

<sensor sensorName="LoanApplicationSensor"
    classname="oracle.tip.pc.services.reports.dca.agents.BpelVariableSensorAgent"
    kind="variable"
    target="$input/payload">
    <variableConfig outputNamespace="http://www.autoloan.com/ns/autoloan"
        outputDataType="loanApplication"/>
</sensor>

```

If you want to monitor faults from the identity service, create a fault sensor, as shown in [Figure 17-4](#).

Figure 17–4 Creating a Fault Sensor



A new entry is created in the `bpel_process_name_sensor.xml` file, as shown in [Example 17–3](#):

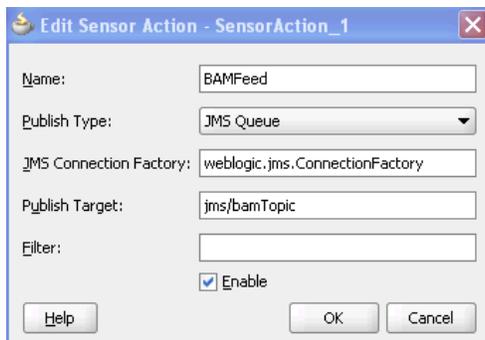
Example 17–3 `bpel_process_name_sensor.xml` file

```
<sensor sensorName="IdentityServiceFault"
  classname="oracle.tip.pc.services.reports.dca.agents.BpelFaultSensorAgent"
  kind="fault"
  target="is:identityServiceFault">
  <faultConfig/>
</sensor>
```

17.2.2 How to Configure Sensor Actions

When you create sensors, you identify the activities, variables, and faults you want to monitor during runtime. If you want to publish the values of the sensors to an endpoint (for example, you want to publish the data of `LoanApplicationSensor` to a JMS queue), then create a sensor action, as shown in [Figure 17–5](#), and associate it with the `LoanApplicationSensor`.

Figure 17–5 Creating a Sensor Action



A new entry is created in the `bpel_process_name_sensorAction.xml` file, as shown in [Example 17–4](#):

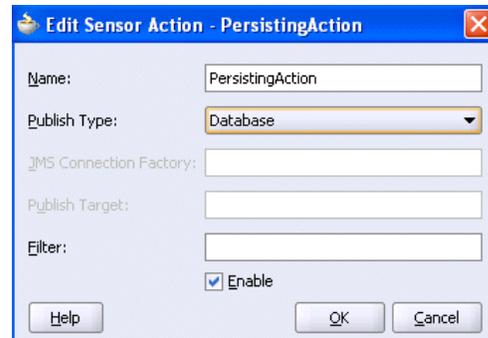
Example 17-4 *bpel_process_name_sensorAction.xml file*

```

<action name="BAMFeed"
  enabled="true"
  publishType="JMSQueue"
  publishTarget="jms/bamTopic">
  <sensorName>LoanApplicationSensor</sensorName>
  <property name="JMSConnectionFactory">
    weblogic.jms.ConnectionFactory
  </property>
</action>

```

If you want to publish the values of `LoanApplicationSensor` and `CreditRatingSensor` to the reports schema in the database, create an additional sensor action, as shown in [Figure 17-6](#), and associate it with both `CreditRatingSensor` and `LoanApplicationSensor`.

Figure 17-6 *Creating an Additional Sensor Action*

A new entry is created in the `bpel_process_name_sensorAction.xml` file, as shown in [Example 17-5](#):

Example 17-5 *bpel_process_name_sensorAction.xml file*

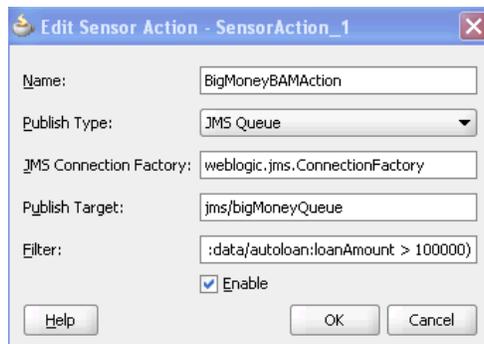
```

<action name="PersistingAction"
  enabled="true"
  publishType="BPELReportsSchema">
  <sensorName>LoanApplicationSensor</sensorName>
  <sensorName>CreditRatingSensor</sensorName>
</action>

```

The data of one sensor can be published to multiple endpoints. In the two preceding code samples, the data of `LoanApplicationSensor` is published to a JMS queue and to the reports schema in the database.

If you want to monitor loan requests for which the loan amount is greater than \$100,000, you can create a sensor action with a filter, as shown in [Figure 17-7](#).

Figure 17–7 Creating a Sensor Action with a Filter

A new entry is created in the `bpel_process_name_sensorAction.xml` file, as shown in [Example 17–6](#):

Example 17–6 `bpel_process_name_sensorAction.xml` file

```
<action name="BigMoneyBAMAction"
  enabled='true'
  filter="boolean(/s:actionData/s:payload
    /s:variableData/s:data
    /autoloan:loanAmount > 100000)"
  publishType="JMSQueue"
  publishTarget="jms/bigMoneyQueue">
  <sensorName>LoanApplicationSensor</sensorName>
  <property name="JMSSConnectionFactory">
    weblogic.jms.ConnectionFactory
  </property>
</action>
```

Notes:

- You must specify all the namespaces that are required to configure an action filter in the sensor action configuration file.
 - You must specify the filter as a boolean XPath expression.
-
-

If you have special requirements for a sensor action that cannot be accomplished by using the built-in publish types (database, JMS queue, JMS topic, and JMS Adapter), then you can create a sensor action with the custom publish type, as shown in [Figure 17–8](#). The name in the **Publish Target** field denotes a fully qualified Java class name that must be implemented.

Figure 17–8 Using the Custom Publish Type

The screenshot shows the 'Create Sensor Action' dialog box with the following fields and values:

- Name: MyAction
- Publish Type: Custom
- JMS Connection Factory: (empty)
- Publish Target: loanflow.MyPublisher
- Filter: (empty)
- Enable: Enable

Buttons: Help, OK, Cancel

17.2.3 How to Publish to Remote Topics and Queues

The JMS queue and JMS topic publish types only publish to local JMS destinations. If you want to publish sensor data to remote topics and queues, use the JMS adapter publish type, as shown in [Figure 17–9](#).

Figure 17–9 Using the JMS Adapter Publish Type

The screenshot shows the 'Create Sensor Action' dialog box with the following fields and values:

- Name: BAMFeed
- Publish Type: JMS Adapter
- JMS Connection Name: weblogic.jms.ConnectionFactory
- Publish Target: java:comp/resource/ojmsdemo/Queues/a
- Filter: (empty)
- Enable: Enable

Buttons: Help, OK, Cancel

In addition to enabling you to publish sensor data to remote topics and queues, the JMS adapter supports a variety of different JMS providers, including:

- Third-party JMS providers such as Tibco JMS, IBM WebSphere MQ JMS, and SonicMQ
- Oracle Enterprise Messaging Service (OEMS) providers such as memory/file and database

If you select the JMS Adapter publish type, you must create an entry in the `weblogic-ra.xml` file, which is updated through the Oracle WebLogic Server Administration Console. Each JMS connection factory (pool) entry created in this console corresponds to one JNDI entry in `weblogic-ra.xml`. Update the Sensor Actions dialog with the chosen JNDI name selected during the creation of the JMS connection factory (pool).

For more information about the JMS adapter, see *Oracle Fusion Middleware User's Guide for Technology Adapters*.

17.2.4 How to Create a Custom Data Publisher

To create a custom data publisher, perform the following steps:

To create a custom data publisher:

1. In the Application Navigator, double-click the BPEL project.

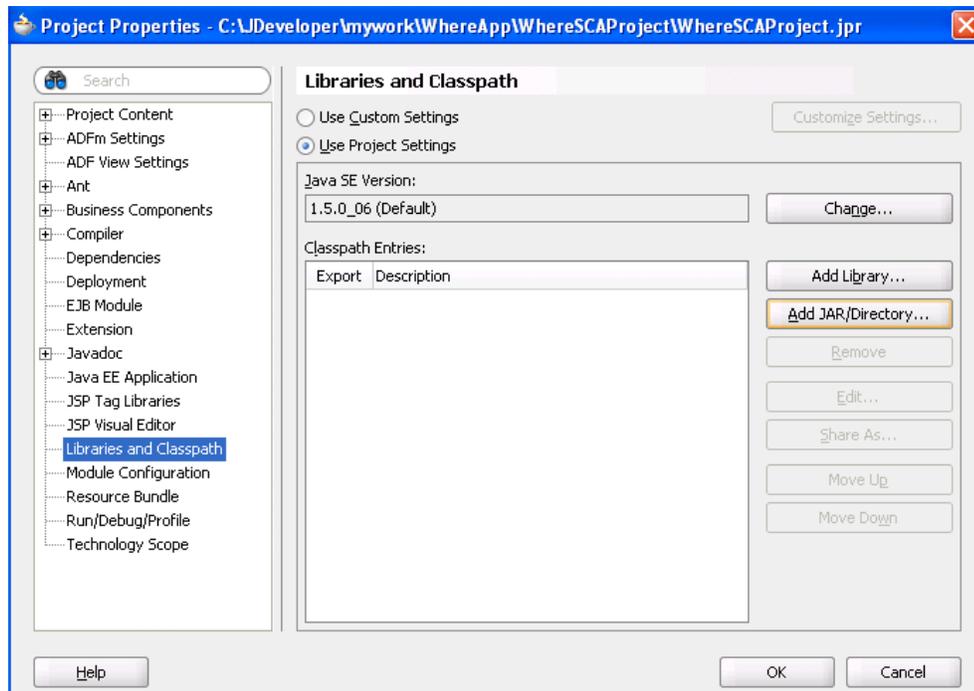
The Project Properties dialog appears.

2. Click **Libraries and Classpath**.

3. Browse and select the following:

`SOA_ORACLE_HOME\lib\java\shared\oracle.soainfra.common\11.1.1\orabpel.jar`

Figure 17–10 Project Properties Dialog



4. Create a new Java class.

The package and class name must match the publish target name of the sensor action.

5. Implement the `com.oracle.bpel.sensor.DataPublisher` interface.

This updates the source file and fills in the methods and import statements of the **DataPublisher** interface.

6. Using the Oracle JDeveloper editor, implement the publish method of the `DataPublisher` interface, as shown in the sample custom data publisher class in [Figure 17–11](#).

Figure 17–11 Custom Data Publisher Class

```

MyPublisher.java
package loanflow;

import com.oracle.bpel.sensor.DataPublisher;

import com.oracle.bpel.sensor.schemas.ITHeaderInfo;
import com.oracle.bpel.sensor.schemas.ITSensorAction;
import com.oracle.bpel.sensor.schemas.ITSensorActionData;
import com.oracle.bpel.sensor.schemas.ITSensorData;

import org.w3c.dom.Element;

public class MyPublisher implements DataPublisher
{
    public MyPublisher()
    {
    }

    public void publish(ITSensorAction action,
        ITSensorActionData actionData,
        Element xml) throws Exception
    {
        ITHeaderInfo header = actionData.getHeader();
        ITSensorData data = actionData.getPayload();

        // Print information on who fired the sensor
        System.out.println("Sensor " + header.getSensor().getSensorName() +
            " fired for BPEL process " + header.getProcessName());

        // Print the sensor data to the console
        System.out.println("Sensor data: " + xml.toString());
    }
}

```

7. Ensure that the class compiles successfully.

The next time that you deploy the BPEL process, the Java class is added to the SOA archive (SAR) and deployed.

Note: Ensure that additional Java libraries needed to implement the data publisher are in the CLASSPATH.

Oracle BPEL Process Manager can execute multiple process instances simultaneously, so ensure that the code in your data publisher is thread safe, or add appropriate synchronization blocks. To guarantee high throughput, do not use shared data objects that require synchronization.

17.2.5 How to Register the Sensors and Sensor Actions in composite.xml

Oracle JDeveloper automatically updates the `composite.xml` file to include appropriate properties for sensors and sensor actions, as shown in [Example 17–7](#):

Example 17–7 composite.xml File

```

<composite name="JMSQFCComposite" applicationName="JMSQueueFilterApp"
  revision="1.0" label="2007-04-02_14-41-31_553" mode="active" state="on">
  <import namespace="http://xmlns.oracle.com/JMSQueueFilter"
    location="JMSQueueFilter.wsdl" importType="wsdl" />
  <service name="client">
    <interface.wsdl interface="http://xmlns.oracle.com/
      JMSQueueFilter#wsdl.interface(JMSQueueFilter)" />
    <binding.ws
      port="http://xmlns.oracle.com/JMSQueueFilter#wsdl.endpoint(client/
        JMSQueueFilter_pt)" />
  </service>
  <component name="JMSQueueFilter">
    <implementation.bpel src="JMSQueueFilter.bpel" />
    <property name="configuration.sensorLocation" type="xs:string"
      many="false">JMSQueueFilter_sensor.xml</property>
    <property name="configuration.sensorActionLocation" type="xs:string"
      many="false">JMSQueueFilter_sensorAction.xml</property>
  </component>
  <wire>
    <source.uri>client</source.uri>
    <target.uri>JMSQueueFilter/client</target.uri>
  </wire>
</composite>

```

You can specify additional properties with `<property name= . . .>`, as shown in [Example 17–7](#).

17.3 Viewing Sensors and Sensor Action Definitions in Oracle Enterprise Manager Fusion Middleware Control Console

The Oracle Enterprise Manager Fusion Middleware Control Console provides support for viewing the metadata of sensors, sensor actions, and the sensor data created as part of the process execution.

For more information, see *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite*.

Notes:

- For this release, BAM sensor actions are not shown in Oracle Enterprise Manager Fusion Middleware Control Console.
 - Only sensors with an associated database sensor action are displayed in Oracle Enterprise Manager Fusion Middleware Control Console. Sensors associated with a JMS queue, JMS topic, remote JMS, or custom sensor action are not displayed
-
-

Part III

Using the Oracle Mediator Service Component

This part describes the components that comprise the Oracle Mediator service component.

This part contains the following chapters:

- [Chapter 18, "Getting Started with Oracle Mediator"](#)
- [Chapter 19, "Creating Mediator Routing Rules"](#)
- [Chapter 20, "Using Mediator Error Handling"](#)
- [Chapter 21, "Working with Multiple Part Messages in Mediator"](#)
- [Chapter 22, "Understanding Message Exchange Patterns of a Mediator"](#)

Getting Started with Oracle Mediator

This chapter provides you an overview of Oracle Mediator (Mediator) and also describes how to create an Oracle Mediator service component.

This chapter includes the following sections:

- [Section 18.1, "Introduction to Oracle Mediator"](#)
- [Section 18.2, "Overview of Mediator Editor Environment"](#)
- [Section 18.3, "Creating a Mediator"](#)
- [Section 18.4, "Generating a WSDL File"](#)
- [Section 18.5, "Specifying Operation or Event Subscription Properties"](#)
- [Section 18.6, "Modifying a Mediator Component"](#)

18.1 Introduction to Oracle Mediator

Oracle Mediator provides a lightweight framework to mediate between various components within a composite application. Mediator converts data to facilitate communication between different interfaces exposed by different components, which are wired together to build a SOA composite application. For example, a Mediator can accept data contained in a text file from an application or service, transform it to a format appropriate for updating a database that serves as a customer repository, and then route and deliver the data to that database.

Oracle Mediator facilitates integration between events and services, where service invocations and events can be mixed and matched. You can use a Mediator component to consume a business event or to receive a service invocation. A Mediator component can evaluate routing rules, perform transformations, validate, and either invoke another service or raise another business event. You can use a Mediator component to handle returned responses, callbacks, faults, and timeouts.

This section provides an overview of Oracle Mediator features:

- **Content-Based and Header-Based Routing**

Oracle Mediator provides support for setting rules based on message payload or message headers. You can select elements or attributes from the message payload or the message header and based on the values, you can specify an action. For example, Mediator receives a file from an application or service containing data about new customers. Based on the country mentioned in the customer's address, you can route and deliver data to the database storing data for that particular country. Similarly, you can route a message based on the message header.

For more information about access header-based routing, see [Section 19.2.2.9, "Access Headers for Filters and Assignments"](#).

- Synchronous and Asynchronous Interactions

Oracle Mediator provides support for synchronous as well as asynchronous request response interaction. In a synchronous interaction, the client requests for a service and then waits for a response to the request. In an asynchronous interaction, the client invokes the service but does not wait for the response. You can specify a timeout period for an asynchronous interaction, which can be used to perform some action, such as raise an event or start a process.

For more information about synchronous and asynchronous interactions, see [Section 19.2.2.3, "Handling Response Messages"](#) and [Chapter 22, "Understanding Message Exchange Patterns of a Mediator"](#).

- Sequential and Parallel Routing of Messages

A routing rule execution type can be either parallel or sequential. You can configure the execution type from Routing Rules panel.

For more information about sequential and parallel routing of messages, see [Section 19.2.2.2, "Specifying Sequential or Parallel Execution"](#).

- Transformations

Oracle Mediator supports data transformation from one XML schema to another. This feature enables data interchange among applications using different schemas. For example, you can transform a comma-delimited file to the database table structure.

For more information about transformations, see [Section 19.2.2.7, "Creating Transformations"](#).

- Validations

Oracle Mediator provides support for validating the incoming message payload by using a Schematron or an XSD file. You can specify Schematron files for each inbound message part and Oracle Mediator can execute Schematron file validations for those parts.

For more information about validations, see [Section 19.2.2.10, "Using Semantic Validation"](#) and <http://www.schematron.com/>.

- Java Callout

Oracle Mediator provides support for Java callout. Java callouts enable the use of Java code, together with regular expressions.

For more information about Java callout, see [Section 19.2.2.11, "Support for Java Callouts"](#).

- Event Handling

An event is a message data sent as a result of occurrence of an activity in a business environment. Oracle Mediator provides support for subscribing to business events or raising business events. You can subscribe to a business event that is raised when a situation of interest occurs. For example, you can subscribe to an event that is raised when a new customer is created and then use this event to start a business process such as sending confirmation email. Similarly, you can raise business events when a situation of interest occurs. For example, raise a customer created event after completing the customer creation process.

For more information about event handling, see [Chapter 44, "Using Business Events and the Event Delivery Network"](#).

- **Dynamic Routing**

Dynamic Routing separates the control logic, which determines the path taken by the process, from the execution of the process. You can create a dynamic routing rule from the Mediator Editor.

For more information about dynamic routing, see [Section 19.2.3, "Creating Dynamic Routing Rules"](#).

- **Error Handling**

Oracle Mediator supports both fault policy-based and manual error handling. A fault policy consists of conditions and actions. Conditions specify the action to be carried out for a particular error condition.

For more information about error handling, see [Chapter 20, "Using Mediator Error Handling"](#).

- **Mediator Echo Support**

Oracle Mediator supports echoing source messages back to the initial caller after any transforms, validations, assignments, or sequencing are performed.

For more information about Mediator echo support, see ["To echo a service:"](#) on page 19-8.

- **Multiple Part Message Support**

Oracle Mediator supports messages consisting of multiple parts. Some Remote Procedure Call (RPC) Web services contain multiple parts in the SOAP message.

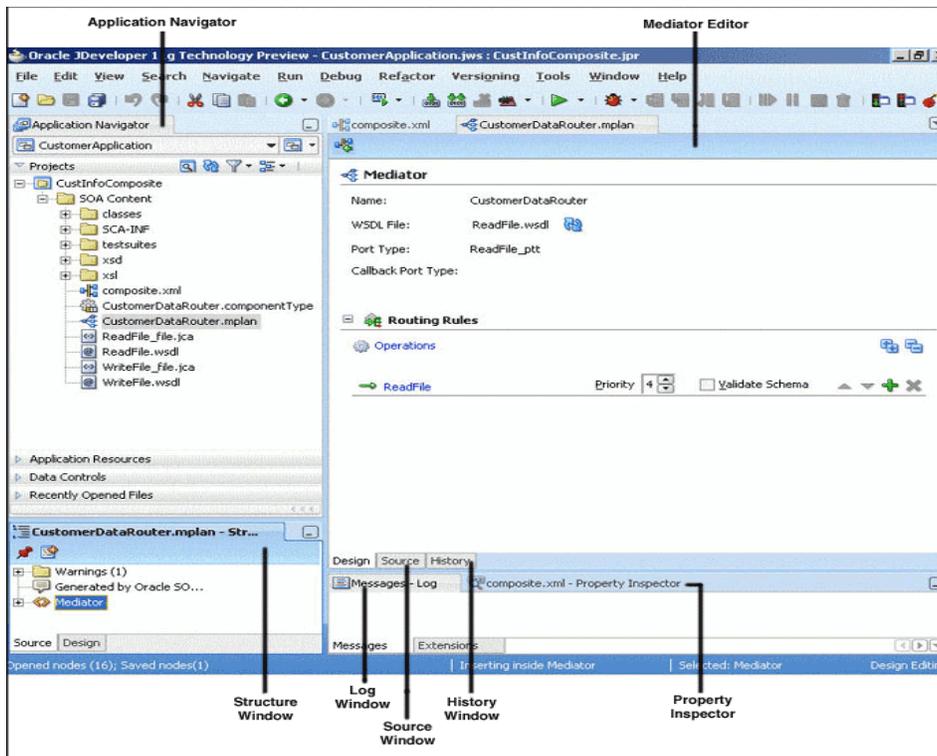
For more information about multiple part message support, see [Chapter 21, "Working with Multiple Part Messages in Mediator"](#).

18.2 Overview of Mediator Editor Environment

You can create a Mediator component in the SOA Composite Application of Oracle JDeveloper and then configure it by using the Mediator Editor. To display the Mediator Editor, double-click the Mediator component in the SOA Composite Editor. For information about the SOA Composite Editor, see [Chapter 4, "Introduction to the Functionality of the SOA Composite Editor"](#).

[Figure 18-1](#) shows the Mediator Editor along with Application Navigator, Structure, and Messages windows.

Figure 18–1 Mediator Editor Window

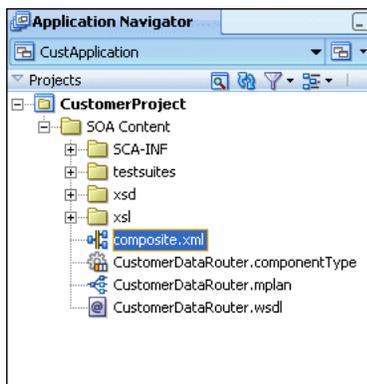


Each section of the view shown in [Figure 18–1](#) enables you to perform specific design and deployment tasks. The following list describes these sections and their functionality:

- **Application Navigator**

The Application Navigator shown in the upper left part of [Figure 18–1](#) displays the Mediator files. [Figure 18–2](#) shows the files that appear under the SOA Content folder when you create a Mediator in a SOA Composite application.

Figure 18–2 Mediator Files in Application Navigator



As shown in [Figure 18–2](#), a SOA Composite application consists of the following Mediator files:

- `Composite.xml`: The file that describes the entire SOA composite application. For information about the `composite.xml` file, see [Chapter 4, "Introduction to the Functionality of the SOA Composite Editor"](#).
- `.componentType`: The `.componentType` file describes the services and references for a service component.
- `.mp1an`: The `.mp1an` file contains Mediator metadata.
- `.wsdl`: A Web Service Description File (WSDL) file specifies how other services call a Mediator. A WSDL file defines the input and output messages and operations of a Mediator.

- **Mediator Editor**

The Mediator Editor, shown in the middle of [Figure 18-1](#), provides a visual view of the Mediator that you have created. This view is displayed when you perform one of the following actions:

- Double-click a Mediator in the SOA Composite Editor.
- Double-click the `.mp1an` file name in the Application Navigator.

- **Source View**

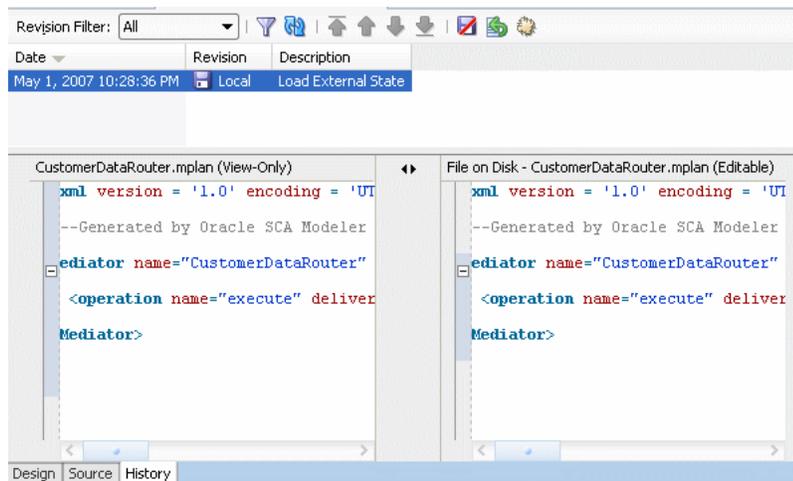
The Source View enables you to view the source code of a Mediator. Click **Source** at the bottom of the Design window shown in [Figure 18-1](#) to view to source code. The code in the source view is immediately updated to reflect the changes in a Mediator.

The following example shows a sample Mediator source code:

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<!--Generated by Oracle SCA Modeler version 1.0 at [4/16/07 10:05 PM].-->
<Mediator name="CustomerDataRouter"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://xmlns.oracle.com/sca/1.0/mediator"/>
```

- **History Window**

The History window enables you to perform tasks as viewing the revision history of a file and viewing read-only and editable versions of a file side-by-side. Click **History** at the bottom of the Design window shown in [Figure 18-1](#) to open the History window. [Figure 18-3](#) shows the History view for a Mediator file.

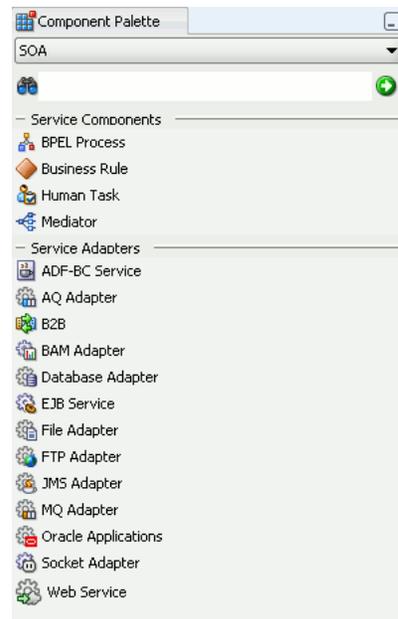
Figure 18–3 History Window

- **Property Inspector**
The Property Inspector shown at the bottom of [Figure 18–1](#) enables you to view details about Mediator properties.
- **Structure Window**
The Structure Window shown in the lower left part of [Figure 18–1](#) provides a structural view of the data of a Mediator.
- **Log Window**
The Log Window displays messages about the status of validation and compilation.

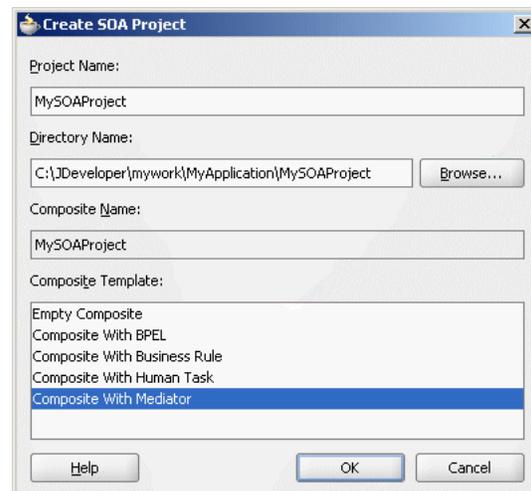
18.3 Creating a Mediator

You can create a Mediator in a SOA Composite application of Oracle JDeveloper by using one of the following methods:

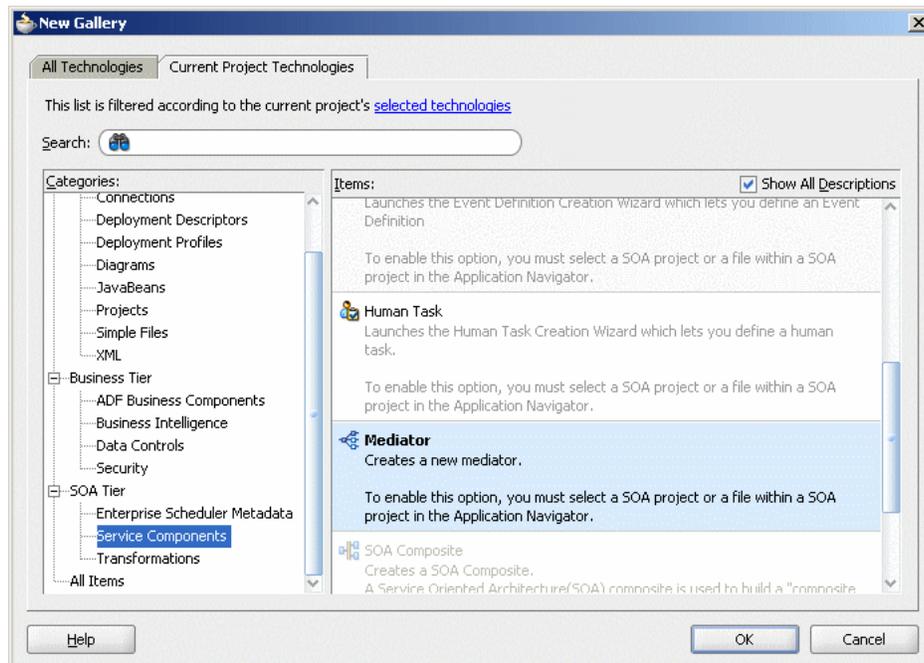
- By dragging and dropping a Mediator from the Component Palette (shown in [Figure 18–4](#)) to SOA Composite Editor.

Figure 18–4 Component Palette with Mediator Service Component

- By selecting Composite with Mediator in the Create SOA Composite dialog or Create SOA Project dialog as shown in [Figure 18–5](#).

Figure 18–5 Composite with Mediator Selection in Create SOA Project Dialog

- By selecting Service Components from the Categories list and Mediator from the Items list in the New Gallery dialog (as shown in [Figure 18–6](#)).

Figure 18–6 New Gallery Dialog with Mediator Service Component

Each method opens the Create Mediator dialog where you specify the name of the Mediator and select a template. A template provides a basic set of default files with which you can begin designing your Mediator.

18.3.1 Creating a Mediator Without Interface Definition

You can create an empty Mediator with no interface definition. This provides you the flexibility to create the SOA components in the order you want. For example, you can create a Mediator first and then create a service or an event that starts the Mediator.

18.3.1.1 How to Create a Mediator with No Interface Definition

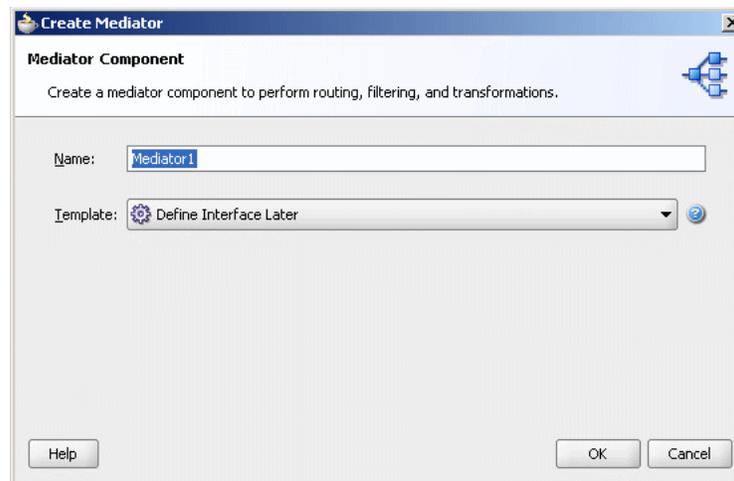
You can create a Mediator with no interface definition by using the Define Interface Later template in the Create Mediator dialog.

To create a Mediator with no interface definition:

1. Drag a **Mediator** component from the SOA list of the Component Palette and drop it in the Components section of the SOA Composite Editor.

The Create Mediator dialog is displayed.

2. In the **Name** field, enter a name for the Mediator component.
3. In the Template list, select **Define Interface Later** as shown in [Figure 18–7](#) and click **OK**.

Figure 18–7 Define Interface Later Template Selection in Create Mediator Dialog

18.3.1.2 How to Define an Interface for a Mediator with no Interface Definition

You can define the interface of a Mediator with no interface definition by subscribing to events or by defining services.

How to Subscribe to Events

You can subscribe to events by selecting the events defined in a `.edl` file.

1. Double-click the Mediator in SOA Composite Editor.
The Mediator Editor is displayed.
2. Click **Add Event Subscription** in the Routing Rules section.
The Subscribed Events dialog is displayed.
3. Click **Add**.
The Event Chooser dialog is displayed.
4. Click **Search** to the right of the **Event definition** field and select an `.edl` file.
The **Event** field is populated with the events defined in the `.edl` file.
5. Select one or more events and click **OK**.
6. In the **Consistency** list, select a level of delivery consistency for the event.
7. In the **Run as Roles** field, you will see `$publisher` as the default security role. You can either retain this value or you can leave this field blank.
8. Double-click the **Filter** field to specify an expression for filtering the event.
9. Click **OK**.

For more information about **Consistency**, **Run as Roles**, and **Filter** fields of an event, see [Section 18.3.6, "Creating a Mediator Component for Event Subscription"](#).

How to Define Services

You can define service for a Mediator with no interface definition in following two ways:

- By connecting the Mediator to a service through a wire in SOA Composite Editor.
- By using the Define Service option in Mediator Editor.

To define services for a Mediator through wire:

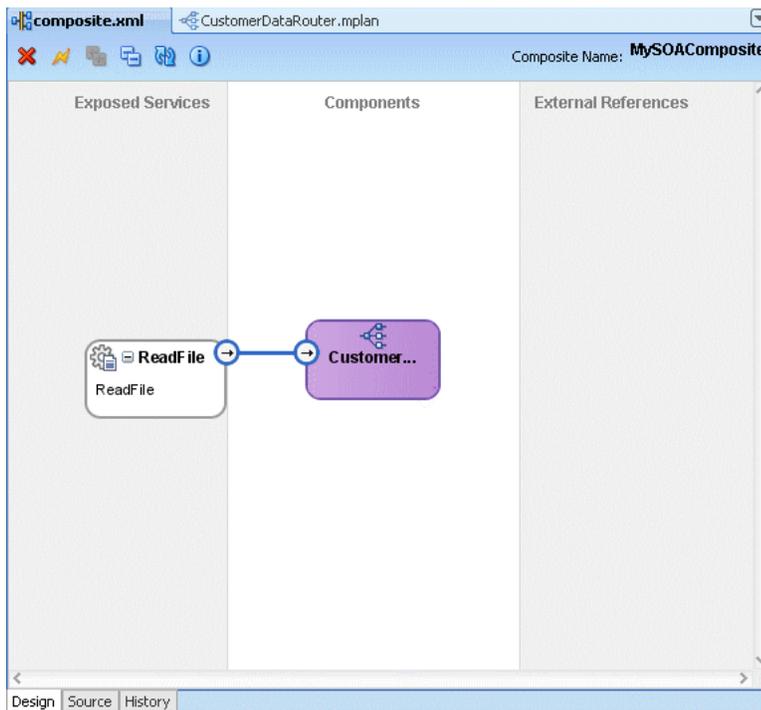
- In SOA Composite Editor, drag a wire from a Mediator to a service.

For more information about wires and how to wire a service component to a service, see [Section 4.2.9, "How to Wire a Service and a Service Component"](#).

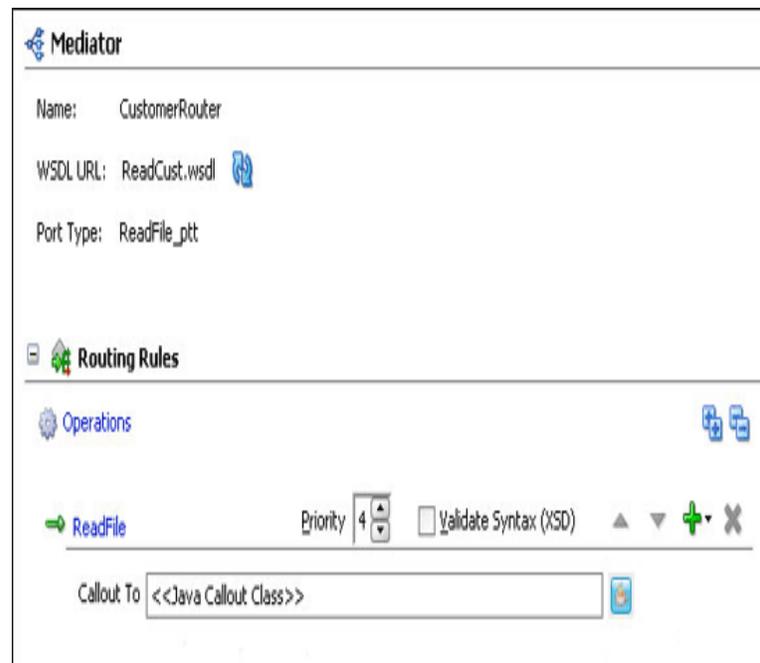
Note: You can also connect a Mediator with defined interface and defined reference to a service through wire. However, to connect Mediator to a service, the interface of the Mediator and the service must match.

The service for a Mediator is automatically defined by using the WSDL file from the wire source. For example, if you connect the `ReadFile` service shown in [Figure 18-8](#) to the `CustomerDataRouter` Mediator, then the `CustomerDataRouter` Mediator automatically inherits the service definition of the `ReadFile` service.

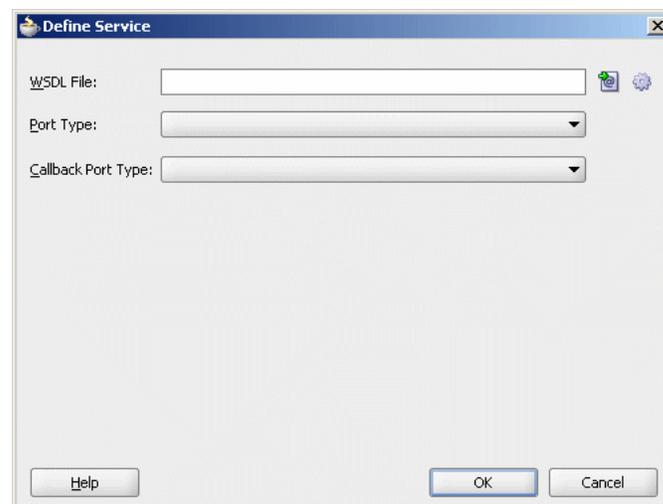
Figure 18-8 Connecting Mediator to a Service



When you double-click the Mediator, the Mediator Editor shown in [Figure 18-9](#) is displayed.

Figure 18–9 Mediator Editor**To define services for a Mediator in Mediator Editor:**

1. Double-click the Mediator in SOA Composite Editor.
The Mediator Editor is displayed.
2. Click **Add** to the right of WSDL File.
The Define Service dialog is displayed, as shown in [Figure 18–10](#).

Figure 18–10 Define Service Dialog

3. Click **Find Existing WSDLs** to use an existing WSDL file or **Generate WSDL From Schema(s)** to create a new WSDL file.

For information about how to generate a WSDL file, see [Section 18.4, "Generating a WSDL File"](#).

4. In the Port type list, select a port.
5. In the Callback Port Type list, select a port for the response message in asynchronous interaction.
6. Click OK.

18.3.2 Creating a Mediator Based on a WSDL File

You can create a Mediator based on an existing WSDL file. A WSDL file describes the interface of a Mediator such as schemas and operations.

18.3.2.1 How to Create a Mediator Based on a WSDL File

You can create a Mediator based on a WSDL file by using the Interface Definition from the WSDL template in the Create Mediator dialog.

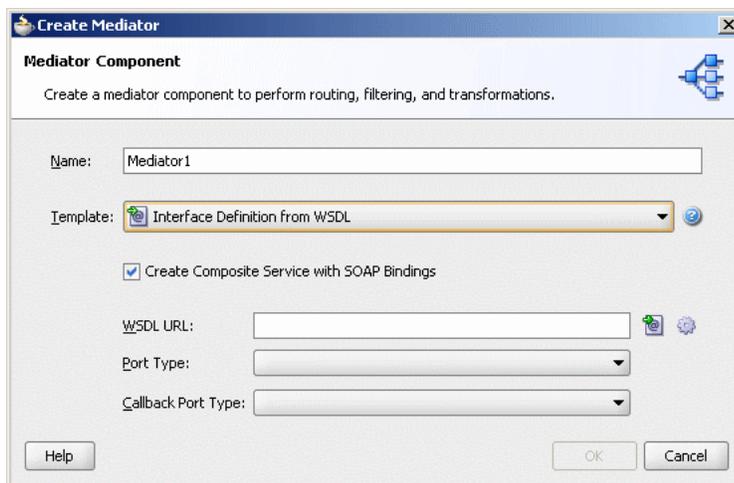
To create a Mediator based on a WSDL File Interface:

1. Drag a **Mediator** component from the SOA list of the Component Palette and drop it in the Components section of the SOA Composite Editor.

The Create Mediator dialog is displayed.

2. In the **Name** field, enter a name for the Mediator component.
3. In the Template list, select **Interface Definition From WSDL** as shown in [Figure 18–11](#).

Figure 18–11 Interface Definition from WSDL Template Selection in Create Mediator Dialog



4. Deselect the Create Composite Service with SOAP Bindings option if you do not want to create an exposed service with SOAP bindings that are automatically connected to your Mediator.
5. In the **WSDL File** field, enter the name of the WSDL file.

You can either use an existing WSDL file or create a new WSDL file. Click **Find Existing WSDL files** to use an existing WSDL file or **Generate WSDL From Schema(s)** to create a new WSDL file.

For more information about these options, refer to [Section 18.4, "Generating a WSDL File"](#).

6. In the Port Type list, select a port. This parses the WSDL file that you specify in the **WSDL File** field to display the list of port types.
7. In the Callback Port Type list, select a callback port. A callback port is the one to which the response message is sent in asynchronous communication.
8. Click **OK**.

18.3.3 Creating a Mediator with One-Way Interface Definition

A Mediator supports one-way interaction. In a one-way interaction, the client sends a message to the service, and the service does not need to reply.

18.3.3.1 How to Create a Mediator with One-Way Interface Definition

You can create a Mediator for a one-way interaction by using the One-Way Interface template in the Create Mediator dialog.

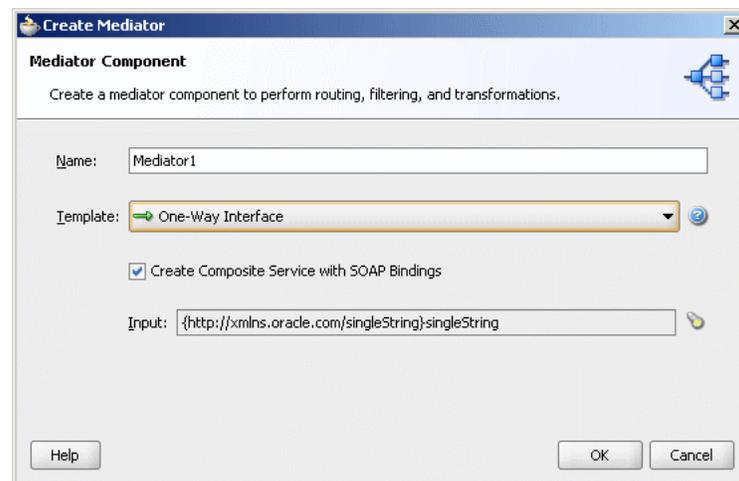
To create a Mediator with one-way interface definition:

1. Drag a **Mediator** component from the SOA list of the Component Palette and drop it in the Components section of the SOA Composite Editor.

The Create Mediator dialog is displayed.

2. In the **Name** field, enter a name for the Mediator component.
3. In the Template list, select **One-Way Interface** as shown in [Figure 18–12](#).

Figure 18–12 One-Way Interface Template Selection in Create Mediator Dialog



4. Deselect the Create Composite Service with SOAP Bindings option if you do not want to create an exposed service with SOAP bindings that are automatically connected to your Mediator component.
5. Click **Search** to the right of the **Input** field to select a schema element for the input message. By default, `singleString` schema element is selected for the input message.

Note: You can use any XSD schema to specify the format of the input document that Mediator will process. For example, you can use the following schema:

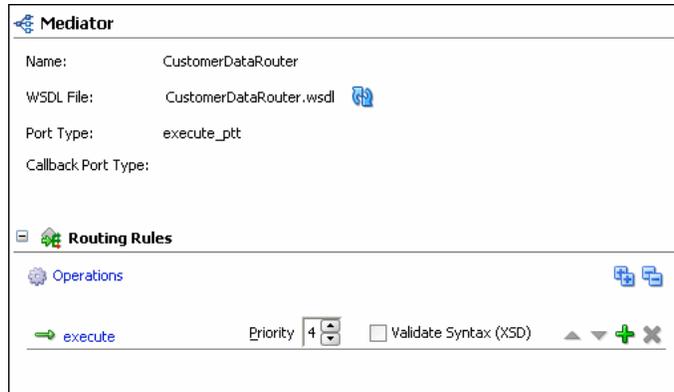
```
<xsd:schema attributeFormDefault="qualified"
  elementFormDefault="qualified"
  targetNamespace="http://samples.otn.com/helloworld"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns="http://samples.otn.com/helloworld">
  <include namespace="http://samples.otn.com/helloworld"
    schemaLocation="helloworld.xsd" />
  <xsd:element name="name1" type="xsd:string" />
  <xsd:element name="result1" type="xsd:string"/>
</xsd:schema>
```

6. Click OK.

18.3.3.2 What Happens When You Create a Mediator Component with One-Way Interface Definition

A Mediator for one-way interaction with port type defined for the input message is created. Figure 18–13 shows how a Mediator created with one-way interface looks like in Mediator Editor. The arrows to the left of the execute operation in Figure 18–15 represent a one-way operation.

Figure 18–13 One-Way Interface Mediator in Mediator Editor



18.3.4 Creating a Mediator with Synchronous Interface Definition

A Mediator supports synchronous request-response interaction. In a synchronous interaction, a client sends a request to a service and receives an immediate response. The client does not proceed further until the response arrives.

18.3.4.1 How to Create a Mediator with Synchronous Interface Definition

You can create a Mediator for synchronous interaction by using the Synchronous Interface template in the Create Mediator dialog.

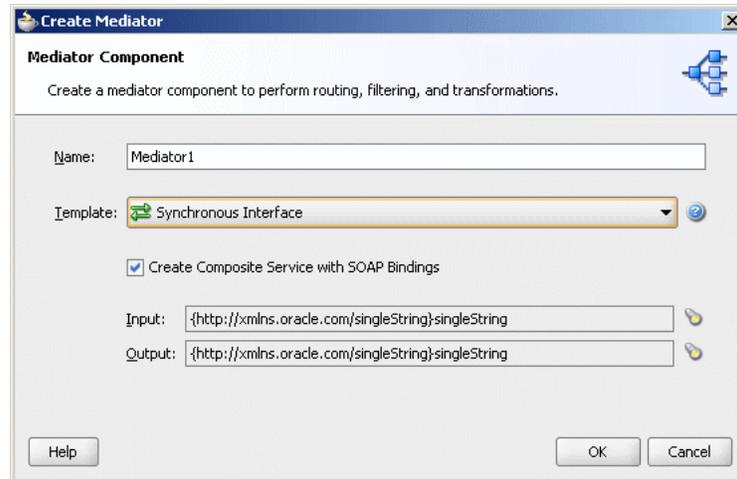
To create a Mediator with synchronous interface definition:

1. Drag a **Mediator** component from the SOA list of the Component Palette and drop it in the Components section of the SOA Composite Editor.

The Create Mediator dialog is displayed.

2. In the **Name** field, enter a name for the Mediator.
3. In the Template list, select **Synchronous Interface** as shown in [Figure 18–14](#).

Figure 18–14 Synchronous Interface Template Selection in Create Mediator Dialog

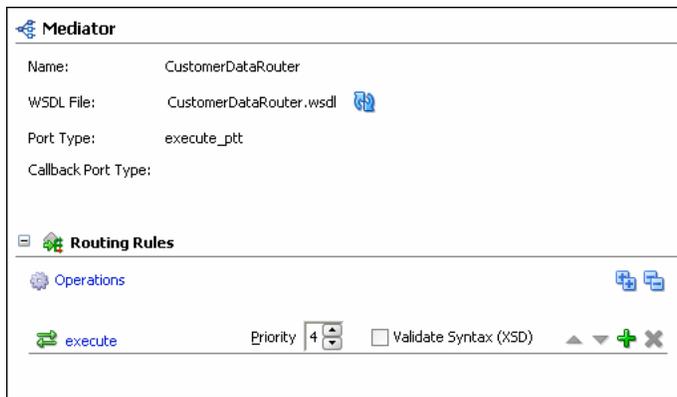


4. Deselect the Create Composite Service with SOAP Bindings option if you do not want to create an exposed service with SOAP bindings that are automatically connected to your Mediator.
5. Click **Search** to the right of the **Input** field to select a schema element for the input message. By default, `singleString` schema element is selected for the input message.
6. Click **Search** to the right of the **Output** field to select a schema element for the output message. By default, the `singleString` schema element is selected for the output message.
7. Click **OK**.

18.3.4.2 What Happens When You Create a Mediator Component with Synchronous Interface Definition

A Mediator with port type defined for the request message is created. In a synchronous interaction, because the response is sent to the same port as request, only one port is defined. [Figure 18–15](#) shows how a Mediator created with synchronous interface appears in Mediator Editor. The arrows to the left of the execute operation in [Figure 18–15](#) represent a synchronous operation.

Figure 18–15 Synchronous Mediator Component in Mediator Editor



18.3.5 Creating a Mediator with Asynchronous Interface Definition

A Mediator supports asynchronous request-response interaction. In an asynchronous interaction, a client sends a request to a service but does not block and wait for a reply.

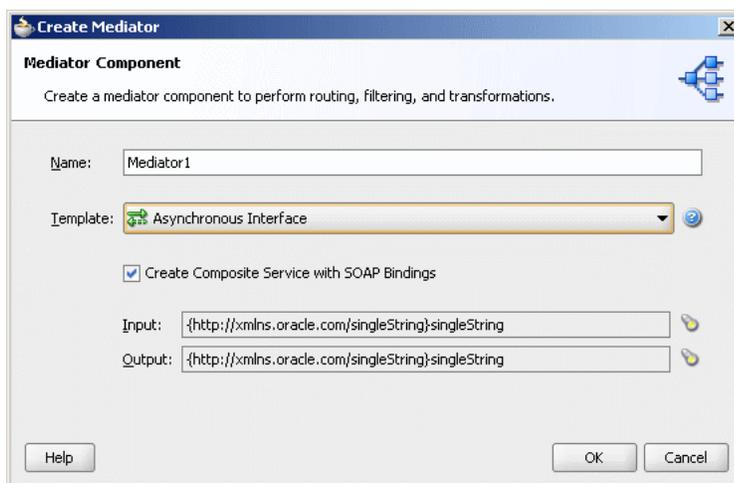
18.3.5.1 How to Create a Mediator with Asynchronous Interface Definition

You can create a Mediator for asynchronous interaction by using the Asynchronous Interface template in the Create Mediator dialog.

To create a Mediator with asynchronous interface definition:

1. Drag a **Mediator** component from the SOA list of the Component Palette and drop it in the Components section of the SOA Composite Editor.
The Create Mediator dialog is displayed.
2. In the **Name** field, enter a name for the Mediator.
3. In the Template list, select **Asynchronous Interface** as shown in [Figure 18–16](#).

Figure 18–16 Asynchronous Interface Template Selection in Create Mediator Dialog



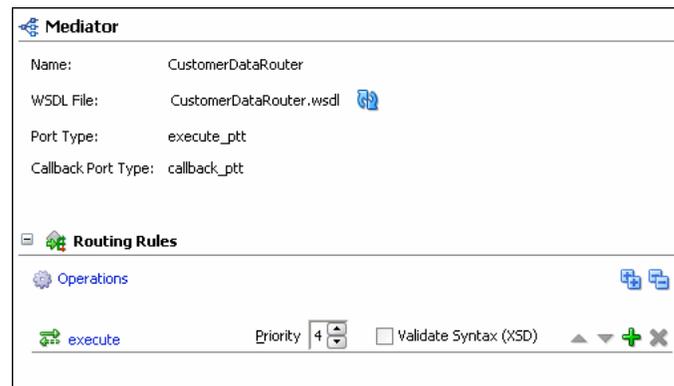
4. Deselect the Create Composite Service with SOAP Bindings option if you do not want to create an exposed service with SOAP bindings that are automatically connected to your Mediator component.

5. Click **Search** to the right of the **Input** field to select a schema element for the input message. By default, `singleString` schema element is selected for the input message.
6. Click **Search** to the right of the **Output** field to select a schema element for the output message. By default, `singleString` schema element is selected for the output message.
7. Click **OK**.

18.3.5.2 What Happens When You Create a Mediator Component with Asynchronous Interface Definition

A Mediator for asynchronous interaction, with port types defined for request and response messages, is created. [Figure 18–17](#) shows how a Mediator created with asynchronous interface looks like in Mediator Editor. The **Port Type** field displays the port on which the request message is sent. The **Callback Port Type** displays the port to which the response is sent. The arrows to the left of the execute operation in [Figure 18–17](#) represent an asynchronous operation.

Figure 18–17 Asynchronous Mediator in Mediator Editor



18.3.6 Creating a Mediator Component for Event Subscription

You can create a Mediator for subscribing to a business event that is raised when a situation of interest occurs. A business event consists of message data sent as the result of an occurrence in a business environment. For information about business events, see [Chapter 44, "Using Business Events and the Event Delivery Network"](#).

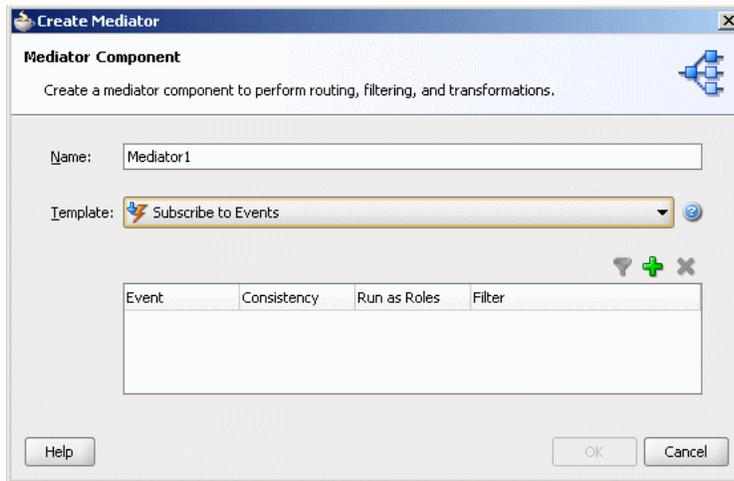
18.3.6.1 How to Create a Mediator for Event Subscription

You can create a Mediator for subscribing to events by using the **Subscribe to Events** template in the Create Mediator dialog.

To create a Mediator for subscribing to events:

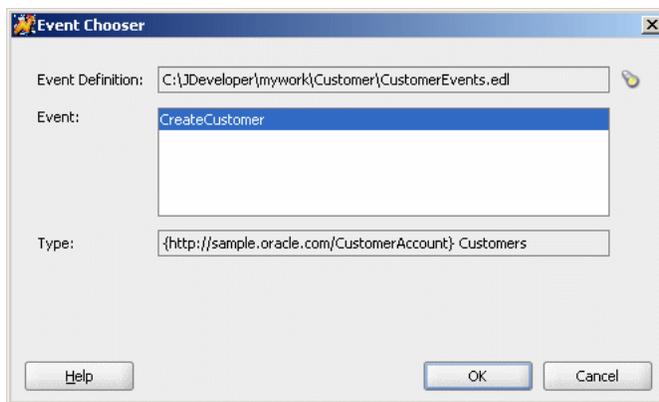
1. Drag a **Mediator** from the SOA list of the Component Palette and drop it in the Components section of the SOA Composite Editor.
The Create Mediator dialog is displayed.
2. In the **Name** field, enter a name for the Mediator component.
3. In the Template list, select **Subscribe to Events** as shown in [Figure 18–18](#).

Figure 18–18 *Subscribe to Events Template Selection in Create Mediator Dialog*



4. Click **Add**.
The Event Chooser dialog is displayed.
5. Click **Search** to the right of the **Event Definition** field.
The SCA Resource Browser dialog is displayed.
6. Select an event definition file (.edl) and click **OK**.
The **Event** field is populated with the events described in the .edl file that you selected. For more information about creating .edl files, see [Chapter 44, "Using Business Events and the Event Delivery Network"](#).
7. Select one or more events in the **Event** field, as shown in [Figure 18–19](#), and click **OK**.

Figure 18–19 *Event Chooser Dialog*

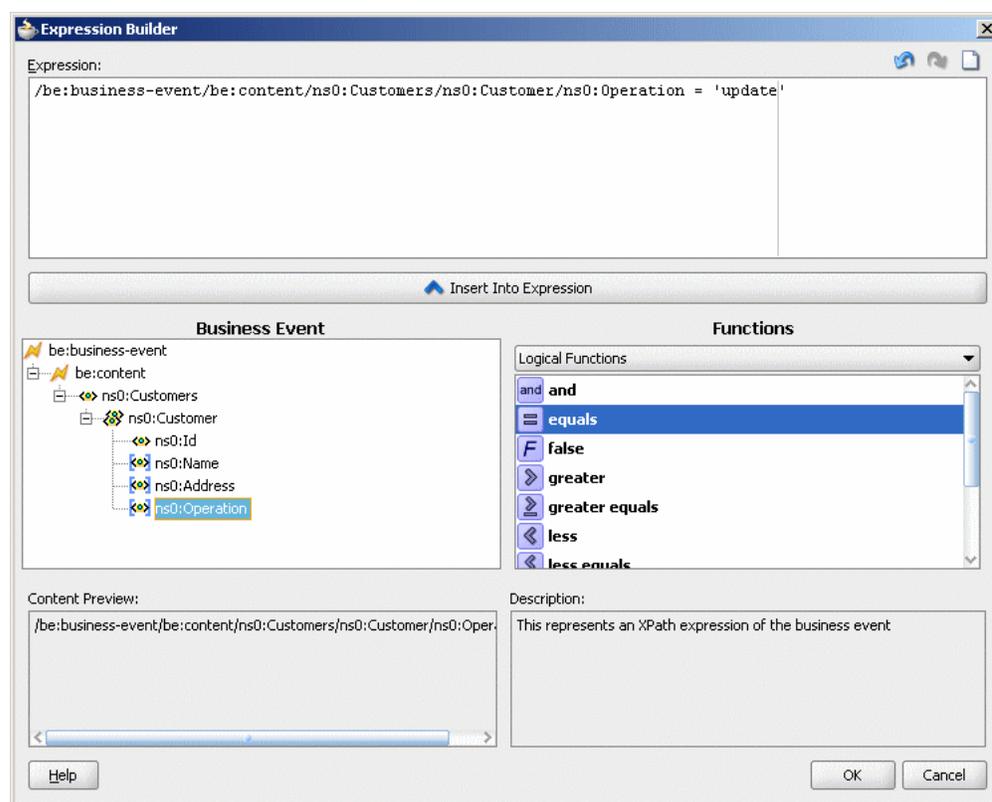


8. Select a level of delivery consistency for the event.
 - one and only one: A global (JTA) transaction is used for event delivery. If the event call fails, the transaction is rolled back and the call is retried a configurable number of times.
 - guaranteed: A local transaction is used to guarantee delivery. There are no retries upon failure.

- immediate: Events are delivered on the same thread and on the same transaction as the caller.
9. In the **Run as Roles** field, enter a security role under which an event subscription is run. By default, event subscription runs under the security of the event publisher `$publisher`. You can either retain this value or leave this field blank.
 10. To filter the event, perform any of the following:
 - Double-click the **Filter** column of the selected event.
 - Select the event and then click the **filter** icon (first icon).
 The Expression Builder dialog is displayed.
 11. In the **Expression** field, enter an XPath expression and click **OK**.

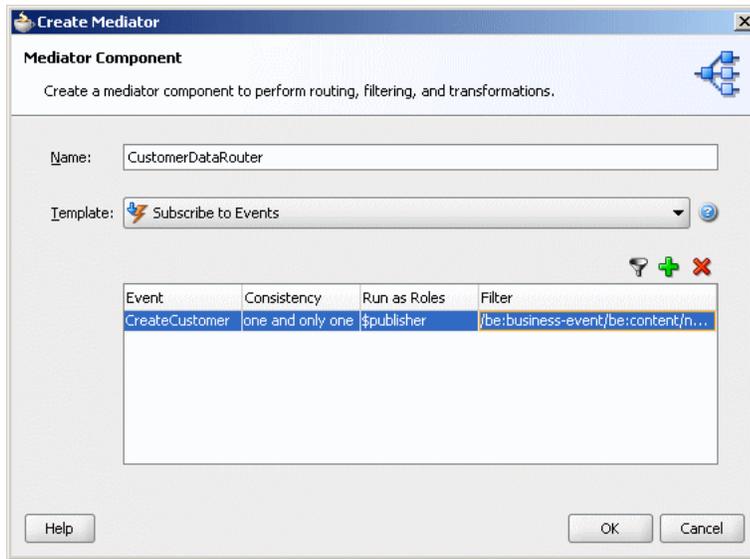
Figure 18–20 shows a sample Expression Builder dialog.

Figure 18–20 Business Event Filter



The Filter column of the Create Mediator dialog is populated as shown in Figure 18–21.

Figure 18–21 Create Mediator Dialog with Filter Expression



12. Click OK.

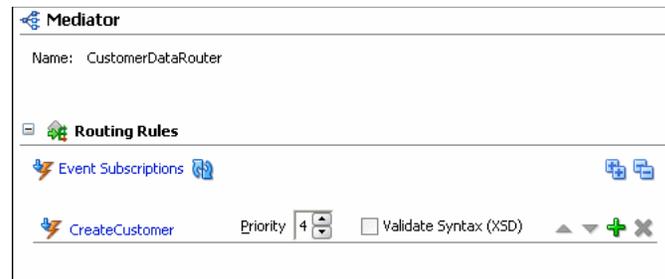
18.3.6.2 What Happens When You Create a Mediator Component for Event Subscription

A Mediator similar to the one shown in [Figure 18–22](#) is created. The icon on the left side of the Mediator indicates that this Mediator is configured for an event subscription.

Figure 18–22 Mediator Component Created with Subscribe to Events Template



When you double-click the Mediator, the Mediator Editor shown in [Figure 18–23](#) is displayed.

Figure 18–23 Mediator Component with Event Subscriptions in Mediator Editor

18.3.7 What You May Need to Know About the Information Available in Mediator User Interface

This section describes the concepts you should know for creating a Mediator component.

18.3.7.1 Mediator Definition

Mediator is a component of Oracle SOA offering that provides mediation capabilities like selective routing, transformation and validation capabilities, along with various message exchange patterns, like synchronous, asynchronous and event publishing or subscription.

For more information about creating a Mediator, see [Section 18.3, "Creating a Mediator"](#).

18.3.7.2 Routing Rule

Routing Rules are mediation logic or execution logic that you define to achieve the requisite mediation. For more information about defining routing rules, see [Section 19.2, "Defining Routing Rules"](#).

You must specify the following for creating a routing rule:

- Operation or Event
 - A Mediator routing rule can be triggered either by a service operation or an event subscription. The service operation can be synchronous, asynchronous or one-way.
- Java Callout
 - Java Callouts are used to perform an external Java logic at various points in the execution of the Mediator.
- Static Routing Rule
 - A Mediator routing rule that is statically defined and is not expected to change depending on the invocation context. In this case, the routing can be echo, routing to another service, or publishing an event.

Static routing rules involve specifying the following:

- Request Handler
 - This defines how Mediator should handle incoming requests.
- Reply Handler
 - This defines how the synchronous response from the called service should be handled by Mediator.

- Fault Handler
This defines how the named or declared faults from the called service should be handled by Mediator.
- Callback Handler
This defines how the asynchronous response/callback from the called service should be handled by Mediator.
- Timeout Handler in Callback
This defines for how much time Mediator should wait for the asynchronous response/callback, before performing timeout handling for the particular asynchronous request.
- Event Publishing and Service Invocation
Event publishing and service invocation call other services or publish an event depending on the configuration of the Handlers.
- Sequential and Parallel Execution
Each routing rule execution can be configured to be either sequential, that is, running in the same thread, or parallel, that is, running in different threads.

Note: For synchronous service invocations, the routing rule should always be sequential.

- Filter Expression
This defines whether a particular routing rule will execute or not. This feature uses XPath Standards and enables selective execution of Mediator routing rule.
- Semantic Validation
This feature enables semantic validation of incoming requests, and also verifies the correctness of data. This feature uses Schematron validation standard.
- Transformation
This feature enables transformation of incoming data to a format that is compliant with called services or published events. This feature is based on XSL transformation standards.
- Assign
This feature enables manipulation of headers and properties for a message to suite the called service.

- Dynamic Routing Rule

A Mediator routing rule that enables you to externalize the routing logic to a Oracle Rules Dictionary, which in turn enables dynamic modification of the routing logic in a routing rule. This feature depends on Decision service and Oracle Rules to obtain the routing logic at runtime.

Note: Oracle recommends using Unicode database with AL32UTF8 as the database character set, for full globalization support in Mediator.

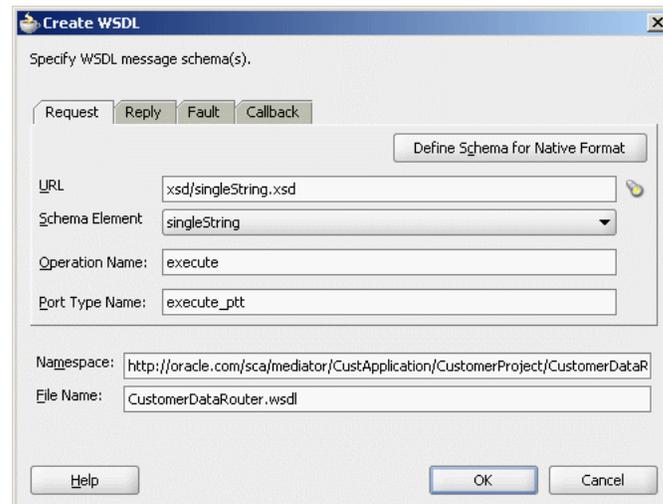
18.4 Generating a WSDL File

You can generate a WSDL file by using either of the following methods:

- By using the Generate WSDL from Schema(s) option that is displayed when you select Interface Definition from WSDL template in the Create Mediator dialog.
- By using the Generate WSDL from Schema(s) option in the Define Service dialog that is displayed while defining services for a Mediator with no interface definition.

Each of these methods opens the Create WSDL dialog shown in [Figure 18–24](#).

Figure 18–24 Create WSDL Dialog



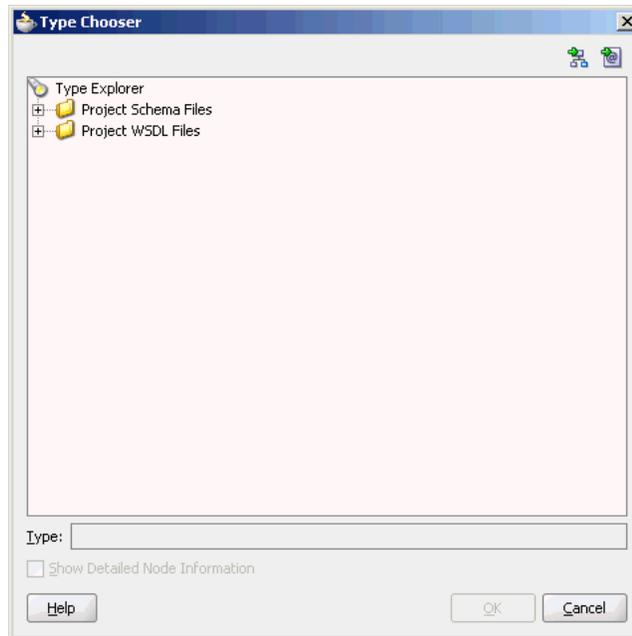
The Create WSDL dialog consists of request, reply, fault, and callback tabs, which you can use to define the schema files for request, reply, fault, and callback messages. You can specify the same or different schema files for the request, response, fault, and callback messages. Minimally, you must specify the schema file for the request message. By default, the `singleString.xsd` file is selected for the request message.

You can generate the WSDL file for a message by using an XML schema definition (XSD) file or by using a sample file.

To generate a WSDL file from an XSD file:

1. In the Request tab of the Create WSDL dialog, click **Search** to access the schema location.

The Type Chooser dialog is displayed, containing a list of the schema files (XSD files).

Figure 18–25 Type Chooser Dialog

2. Expand the Project Schema Files and Project WSDL Files nodes to locate the schema that you want to use.

You can also import a schema XSD file or WSDL file into a project by using the Import Schema File or Import WSDL icons, respectively.

Note: If you want to use a schema XSD file that resides on your local file system, then ensure that the XSD file, and any XSD files that it imports, all reside in the Oracle JDeveloper project directory.

After you specify a file, Oracle JDeveloper parses it to determine the defined schema elements and displays them in a list, from which you can make a selection.

3. Select the root element of the XSD file and click **OK**.
4. In the **Operation Name** field, enter the operation name. For example:
executeQuery

Oracle JDeveloper converts the specified operation into an operation element in the WSDL file.

Note: Spaces are not allowed in an Operation name.

5. In the **Port Type Name** field, enter the port name.
6. In the **Namespace** field, enter a namespace or accept the current value.
For example: `http://oracle.com/esb/namespaces/Mediator`
The namespace that you specify is defined as the `tns` namespace in the WSDL file.
7. In the **Reply** tab, if entering any information, click **Search** to access a schema and then select a schema element.

The Reply tab enables you to specify the schema for a response message in synchronous communication.

8. In the **Fault** tab, if entering any information, click **Search** to access a schema location and then select a schema element. You cannot specify a fault message schema, unless you also specify a response.
9. In the **Callback** tab, if entering any information, click **Search** to access a schema and then select a schema element.

The Callback tab enables you to specify the schema for a response message in asynchronous communication.

10. In the **Operation Name** field, enter the operation name.

For example: `returnQuery`

11. In the **Port Type Name** field, enter the port name to which the response will be sent.
12. Click **OK**.

Generating the WSDL File Based on a Sample File

You can generate a WSDL file from 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. You can use the Native Format Builder wizard to generate a WSDL file based on a sample file. The Native Format Builder wizard is displayed when you click **Define Schema for Native Format** in the request, response, fault, and callback tabs of the Create WSDL dialog. A WSDL file is generated after you complete the wizard.

For information about the Native Format Builder wizard, see *Oracle Fusion Middleware User's Guide for Technology Adapters*.

18.5 Specifying Operation or Event Subscription Properties

After creating a Mediator, you can use the Mediator Editor to specify the Validate Syntax (XSD) property of an operation or event subscription. You can select this option to validate the schemas of the inbound messages. By default, validate schema is set to `false`.

18.6 Modifying a Mediator Component

You can modify the operations or event subscriptions of a Mediator by using the Mediator Editor.

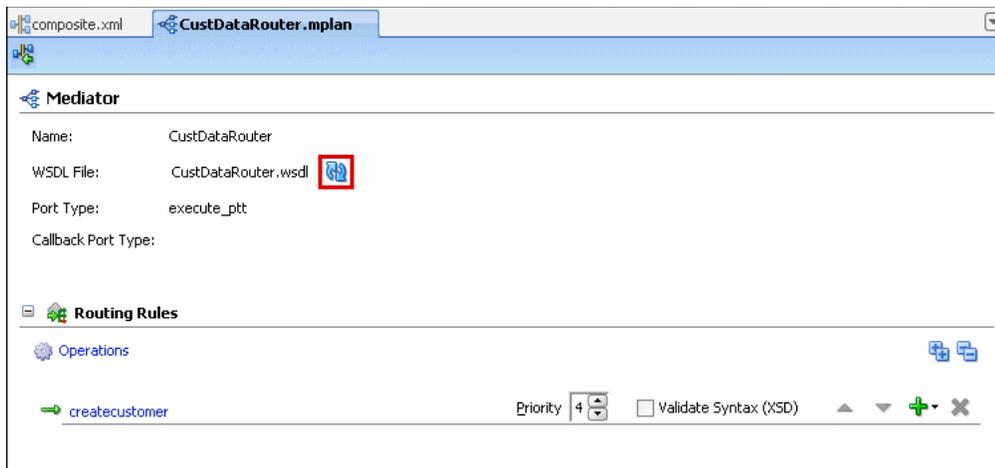
18.6.1 Modifying Operations

You can modify a Mediator WSDL file by adding or deleting operations. After modifying the WSDL file, you can use the Refresh WSDL dialog to synchronize the changes.

To modify the operations of a Mediator:

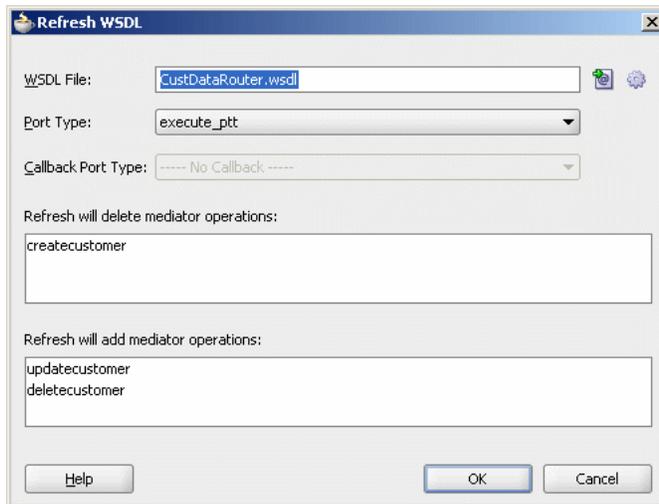
1. In Mediator Editor, click the **Refresh Operations From WSDL** icon (shown highlighted in [Figure 18-26](#)) to the right of the **WSDL File** field.

Figure 18–26 The Refresh Operations From WSDL Icon



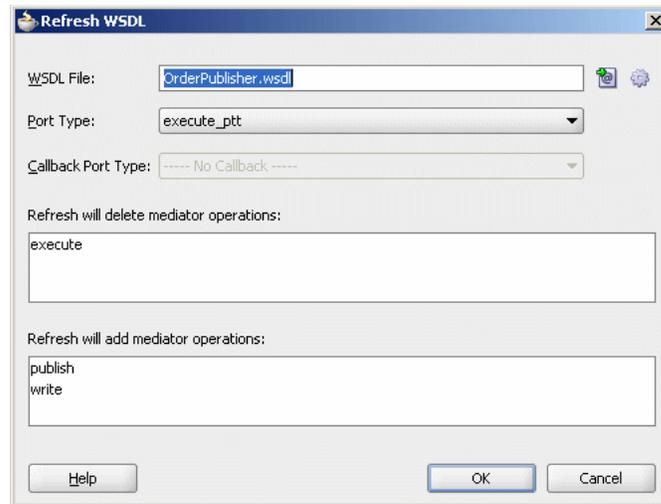
The Refresh WSDL dialog is displayed. If you have made any modifications to the WSDL file, then the Refresh WSDL dialog lists all the operations that will be deleted or added. The Refresh will delete **Mediator operation** field lists all the operation that have been removed from the WSDL file. The Refresh will add **Mediator operation** field lists all the new operation that have been added in the WSDL file. [Figure 18–27](#) displays a Refresh WSDL dialog.

Figure 18–27 Refresh WSDL Dialog



2. To specify a different WSDL file, click **Find Existing WSDLs** to use an existing WSDL file or **Generate WSDL From Schema(s)** to create a new WSDL file.

The Refresh WSDL dialog is updated based on the operations defined in the specified WSDL file as shown in [Figure 18–28](#).

Figure 18–28 Refresh WSDL Dialog with Updated Operations

3. Click **OK**.
4. From the **File** menu, select **Save All**.

18.6.2 Modifying Event Subscriptions

You can subscribe to new events, modify the existing event subscriptions, and unsubscribe from subscribed events by using the Manage Event Subscriptions option in Mediator Editor.

To modify events subscription of a Mediator:

1. In Mediator Editor, click the **Manage Event Subscriptions** icon to the right of the Event Subscriptions.

The Subscribed Events dialog is displayed as shown in [Figure 18–29](#).

Figure 18–29 The Subscribed Events Dialog

2. You can perform any of the following functions:
 - Subscribe to a new event.
 - Unsubscribe from an event.
 - Modify or specify the filter criteria for an event.

- Modify the Consistency or Run as Roles properties of an event subscription.
For more information about **Consistency**, **Run as Roles**, and **Filter** fields of an event, see [Section 18.3.6, "Creating a Mediator Component for Event Subscription"](#).
3. Click **OK**.
 4. From the **File** menu, select **Save All**.

Creating Mediator Routing Rules

This chapter provides an overview of routing rules and describes how to specify routing rules for an Oracle Mediator (Mediator) service component.

This chapter includes the following sections:

- [Section 19.1, "Introduction to Routing Rules"](#)
- [Section 19.2, "Defining Routing Rules"](#)
- [Section 19.3, "Creating a Mediator for Routing Messages"](#)
- [Section 19.4, "Creating Asynchronous Request Response Using Mediator"](#)

19.1 Introduction to Routing Rules

Oracle Mediator enables you to route data between service consumers and service providers. As the data flows from service to service, it must be transformed. These two tasks, routing and transformations, are the core responsibilities of the Mediator. You can use the routing rules to specify how a message processed by a Mediator reaches its next destination. Routing rules specify where a Mediator sends the message, how it sends it, and what changes should be made to the message structure before sending it to the target service.

Routing rules can be of the following two types:

- **Static Routing Rules**
A Mediator routing rule that is statically defined and is not expected to change depending on the invocation context.
- **Dynamic Routing Rules**
A Mediator routing rule that enables you to externalize the routing logic to a Oracle Rules Dictionary, which in turn enables dynamic modification of the routing logic in a routing rule.

For more information on these routing rules, refer to [Section 19.2.2, "Creating Static Routing Rules"](#) and [Section 19.2.3, "Creating Dynamic Routing Rules"](#).

19.2 Defining Routing Rules

Routing rules can be defined only for a Mediator with defined interface. For more information on how to define an interface, refer to [Section 18.3.1.2, "How to Define an Interface for a Mediator with no Interface Definition"](#).

This section includes the following sections:

- [Section 19.2.1, "Using the Routing Rules Panel"](#)
- [Section 19.2.2, "Creating Static Routing Rules"](#)
- [Section 19.2.3, "Creating Dynamic Routing Rules"](#)

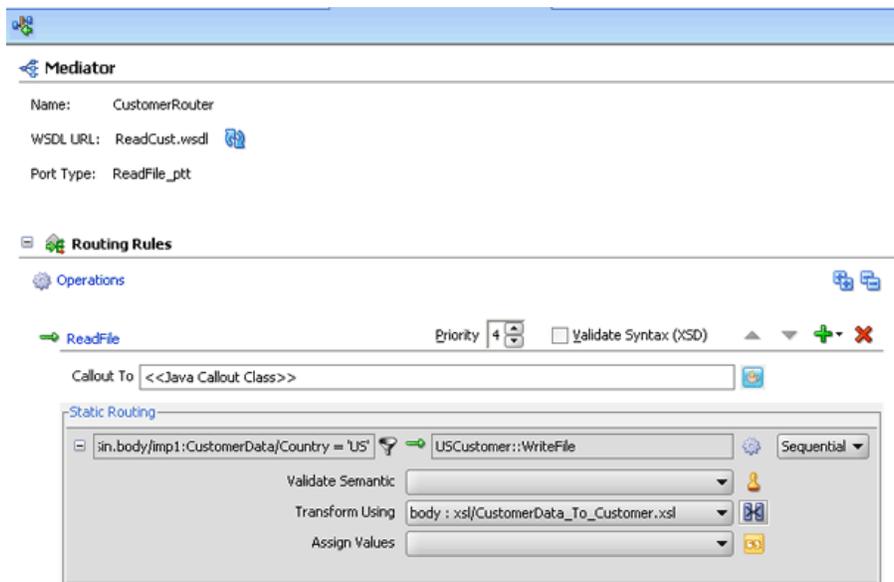
19.2.1 Using the Routing Rules Panel

You can define the routing rules by using the Routing Rules panel of the Mediator Editor. You can access the Mediator Editor by using any one of the following methods:

- From the SOA Composite Editor:
 - a. Double-click the icon that represents the Mediator for which you want to specify the routing rules.
 - b. Click the **Plus (+)** icon next to the Routing Rules panel.
- From the Applications Navigator:
 - a. In the Applications Navigator, expand the SOA project, followed by the SOA Content folder.
 - b. In the SOA Content folder, double-click the name of the Mediator for which you want to specify the routing rules. The Mediator file has `mp1` extension.
 - c. Click the **Plus (+)** icon next to the Routing Rules panel.

Figure 19–1 shows the Mediator Editor with Routing Rules panel.

Figure 19–1 Mediator Editor- Routing Rules Panel



The icons in the Routing Rules panel are summarized in Figure 19–2.

Figure 19–2 Routing Rule Panel Icons

19.2.2 Creating Static Routing Rules

When you configure static routing rules, you can specify the following details:

- **Target service**
Specifies the service to which the message should be sent. See [Section 19.2.2.1, "Specifying Mediator Services or Events"](#) for more information about how to invoke a target service.
- **Execution type**
Specifies the way in which routing rules are executed. You can specify either of the following execution types: sequential or parallel.
See [Section 19.2.2.2, "Specifying Sequential or Parallel Execution"](#) for information about how to specify an execution type.
- **Reply, callback, and fault handlers**
Specify how to handle synchronous reply, callback, and fault messages. See [Section 19.2.2.3, "Handling Response Messages"](#) and [Section 19.2.2.5, "Handling Faults"](#) for information about synchronous reply, callback, and fault messages handling.
- **Filter expression**
Specifies the filter expression to be applied. A filter expression specifies that the contents (payload or headers) of a message be analyzed before any service is invoked. For example, you might apply a filter expression that specifies that a service be invoked only if the message includes a customer ID, or if the value for that customer ID matches a certain pattern. See [Section 19.2.2.6, "Specifying Expression for Filtering Messages"](#) for information about how to specify filter expressions.
- **Transformations**
Specify the transformation to be applied. You can use transformation to set a value on the target payload. You can perform transformation by using mappings or by assigning values.

The XSLT mapper enables you to define transformations that apply to the whole message body, to convert messages from one XML schema to another. The Assign

dialog, on the other hand, works on individual fields. Using this dialog, you can assign values from the message (payload, headers), from constant, or from various system properties, such as the properties of an adapter present in the data path. See [Section 19.2.2.7, "Creating Transformations"](#) and [Section 19.2.2.8, "Assigning Values"](#) for information about how to create transformations.

- **Accessing Header Variables from Expressions**
Detects any SOAP headers that are used in building the expression for the current routing rule operation. See [Section 19.2.2.9, "Access Headers for Filters and Assignments"](#) and [Section 19.2.2.9.2, "Manual Expression Building for Accessing Properties for Filters and Assignments"](#) for information about how to access headers for filters and transformations.
- **Schematron based validations**
Specify the Schematron files for validating different parts of an inbound message. See [Section 19.2.2.10, "Using Semantic Validation"](#) for information about how to perform Schematron based validations.
- **Java callout**
Invokes custom Java class callouts. It enables the use of regular expressions together with Java code, when regular expressions do not suffice. See [Section 19.2.2.11, "Support for Java Callouts"](#) for information about how to use Java callouts.
- **User-defined extension functions**
These are your own set of functions that can be used by the XSL Mapper. See [Section 19.2.2.6.1, "Using User-Defined Extension Functions"](#) for information about how to use user-defined extension functions.

The various types of static routing rules that can be defined for a service or event subscription are the following:

- [Section 19.2.2.1, "Specifying Mediator Services or Events"](#)
- [Section 19.2.2.2, "Specifying Sequential or Parallel Execution"](#)
- [Section 19.2.2.3, "Handling Response Messages"](#)
- [Section 19.2.2.4, "Handling Multiple Callbacks"](#)
- [Section 19.2.2.5, "Handling Faults"](#)
- [Section 19.2.2.6, "Specifying Expression for Filtering Messages"](#)
- [Section 19.2.2.7, "Creating Transformations"](#)
- [Section 19.2.2.8, "Assigning Values"](#)
- [Section 19.2.2.9, "Access Headers for Filters and Assignments"](#)
- [Section 19.2.2.10, "Using Semantic Validation"](#)
- [Section 19.2.2.11, "Support for Java Callouts"](#)

19.2.2.1 Specifying Mediator Services or Events

After creating the Mediator, you associate it to inbound service operations or event subscriptions and specify the targets of the Mediator. Targets are outbound service operations or event publishing. A target specifies the next service or event to which a Mediator should send the message and what service operation is to be invoked. You can specify a service or an event as target type.

You can also echo source messages back to the initial caller after any transformations, validations, assignments, or sequencing are performed. Whether the echo will be synchronous or asynchronous, depends on the WSDL file of the caller. The echo option is only available for inbound service operations and is not available for event subscriptions.

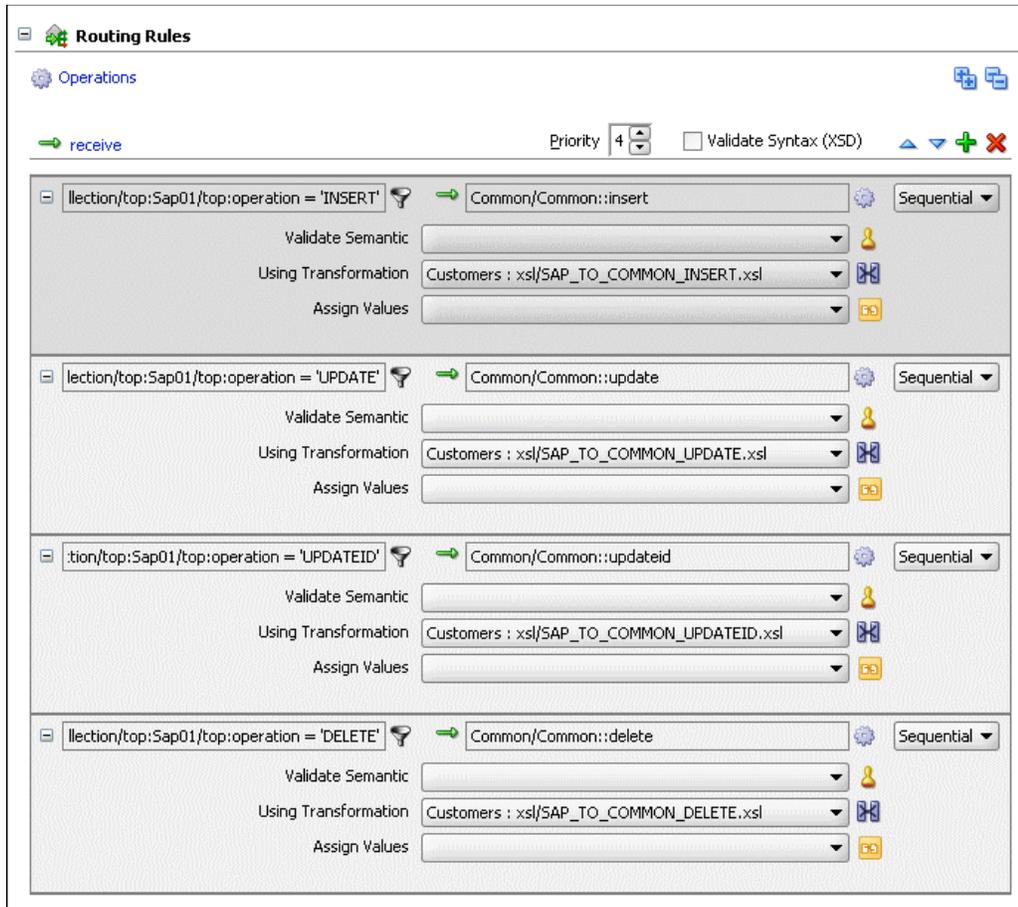
The purpose of the echo option is to expose all the Mediator functionality as a callable service, without having to route to any other service. For example, you can call a Mediator to perform a transformation, a validation, or an assignment, and then echo the Mediator back to your application, without routing anywhere else.

You can specify multiple routings for an inbound operation or event. Each routing is mapped to one target service invocation or event. Therefore, to specify multiple service invocations or raise multiple events, you must specify one routing rule for each target. For example, a message payload, you want to invoke an operation from the following operations defined in a service:

- insert
- update
- updateid
- delete

You must create four routings, one for each operation. Later, when you specify a filter expression, you can specify which target and operation is applied to each message instance, on the basis of the message payload as shown in [Figure 19-3](#).

Figure 19–3 Multiple Routings for an Inbound Operation



To invoke a service:

1. In the Routing Rules panel, click **Add**.

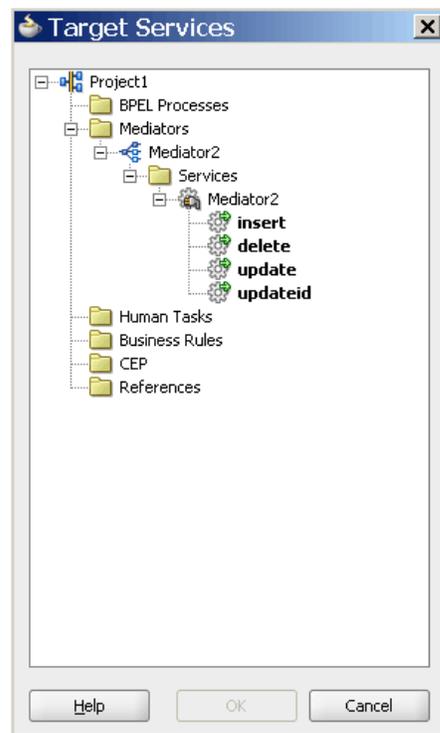
The Target Type dialog is displayed as shown in [Figure 19–4](#).

Figure 19–4 Target Type Dialog



2. Click **Service**.
3. In the Target Services dialog, navigate to, and then select an operation provided by a service, as shown in [Figure 19–5](#).

Figure 19–5 Target Services Dialog



Note: A service can consist of multiple operations as shown in Figure 19–5.

4. Click **OK**.

To raise an event:

1. In the Routing Rules panel, click **Add**.

The Target Type dialog is displayed as shown in Figure 19–6.

2. Click **Event**.

The Event Chooser dialog is displayed.

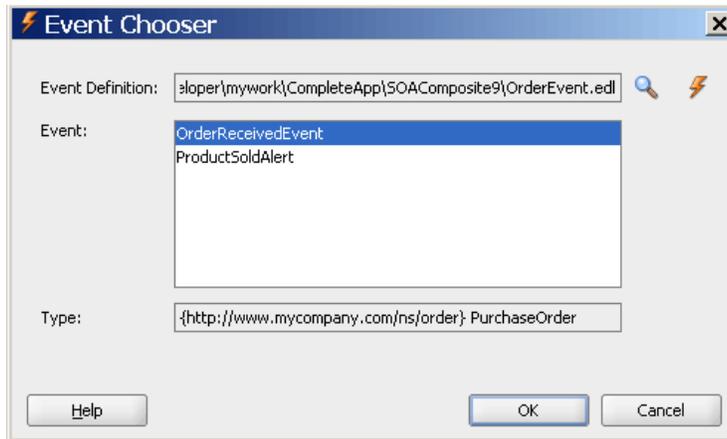
3. Click **Search** to the right of **Event Definition** field.

The SCA Resource Browser dialog is displayed.

4. Select an event file and click **OK**.

The **Event** field is populated with the events defined in the selected file as shown in Figure 19–6.

Figure 19–6 Event Chooser Dialog



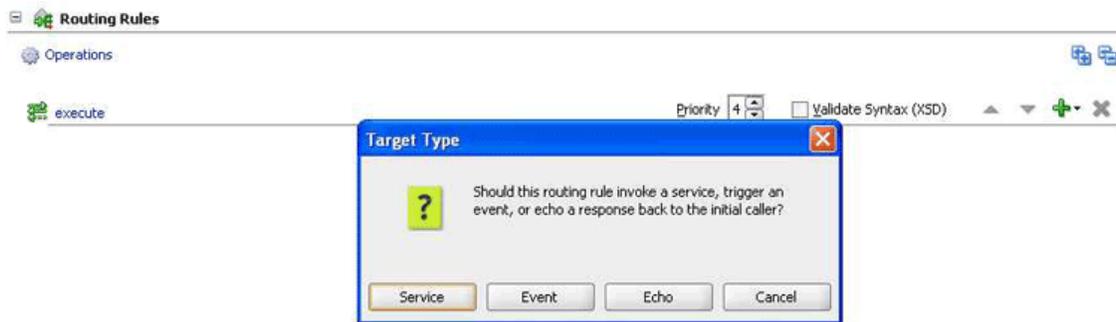
5. Select an event.
6. Click **OK**.

To echo a service:

1. In the Routing Rules panel, click **Add**.

The Target Type dialog is displayed as shown in the following figure:

Figure 19–7 Target Type Dialog

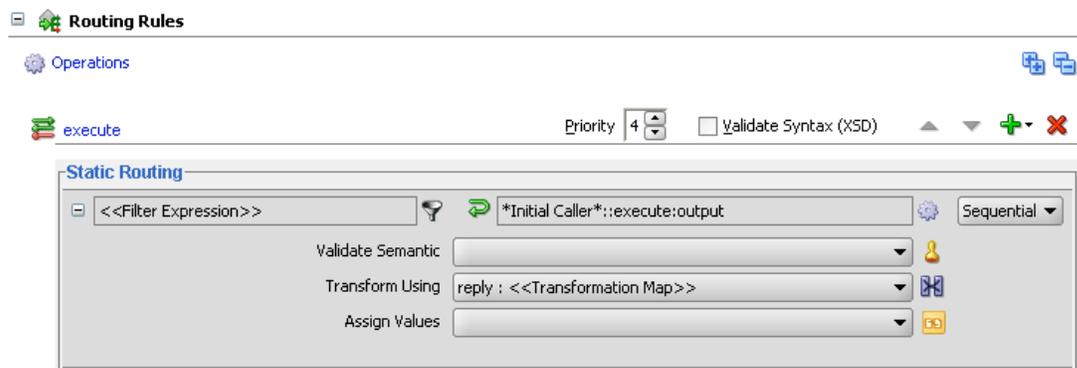


2. Click **Echo**.

The following figure shows a routing rule with a synchronous echo:

Note: An asynchronous echo has an icon with a dotted line on the return.

Figure 19–8 Sample Mediator Supporting Echo Operation



Restrictions on Using Echo Option

The echo option has the following limitations:

- The echo option is supported only with the Mediator interfaces having the following types of WSDL files:
 - Request/Reply
 - Request/Reply/Fault
 - Request/Callback

Note: The echo option is not available for Mediator interfaces having Request/Reply/Fault/Callback WSDL Files.

- The echo option is available for synchronous operations like Request/Reply and Request/Reply/Fault.

Note: The echo option is available for the synchronous operations only when the routing rule is sequential because parallel routing rules are not supported for Mediators with synchronous operations.

- For synchronous operations, having a conditional filter set, the echo option does not return any response to the caller, when the filter condition is set to `false`. Instead, a `null` response is returned to the caller.
- The echo option is available for asynchronous operations *only if* the Mediator interface has a callback operation. In this case, the echo is run on a separate thread.

Note: The asynchronous echo option is available only when the routing rule is parallel. To use the echo option, then sequential routing rules are not supported for Mediators with asynchronous operations.

19.2.2.2 Specifying Sequential or Parallel Execution

You can specify execution type for a routing rule. A routing rule execution type can be parallel or sequential. To specify an execution type for a routing rule, select Sequential or Parallel execution type from the Routing Rules panel.

This section describes the following sections:

- [Basic Principles of Sequential Routing Rules](#)
- [Basic Principles of Parallel Routing Rules](#)

Basic Principles of Sequential Routing Rules

- In sequential execution, routings are evaluated and actions are performed sequentially. Sequential routings are evaluated in the same thread and transaction as the caller.
- Mediator always enlists itself into the global transaction propagated through the thread that is processing the incoming message. For example, if an inbound JCA adapter invokes a Mediator, then the Mediator enlists itself to the transaction that the JCA adapter has initiated.
- Mediator propagates the transaction through the same thread as the target components, while executing the sequential routing rules.
- Mediator never commits or rolls back transactions propagated by external entities.
- Mediator manages the transaction only if the thread-invoking Mediator does not have an active transaction already. For example, if Mediator is invoked from inbound SOAP services, then Mediator starts a transaction, and commits or rolls back the transaction depending on success and failure.

Basic Principles of Parallel Routing Rules

- In parallel execution, routings are queued and evaluated in parallel in different threads.
- A new transaction is initiated by the Mediator for processing each parallel rule. The initiated transaction ends with an enqueue to the Mediator parallel message dehydration store.

For example, if a Mediator component has one parallel routing rule, then one message is enqueued on the Mediator parallel message dehydration store. Then, the parallel message dispatcher to the store initiates a transaction, reads the message from the Database store and invokes the target component or service of this routing rule. This transaction initiated by the listener thread is a completely new transaction and is propagated to the target components.

Note: Dehydrating of messages means storing the incoming messages in database for parallel routing rules, so that they can be processed later by worker threads.

- In parallel execution, Mediator commits or rolls back transactions because it is the initiator of these transactions.

If an operation or event has both sequential and parallel routing rules, first sequential routing rules are evaluated and actions are performed, and then parallel routings are queued for parallel execution.

Note: Mediator does not support parallel execution of routing rules for Mediators with synchronous interface.

19.2.2.3 Handling Response Messages

You can specify how to handle the response messages in synchronous and asynchronous interactions. In case of synchronous interactions, you can specify the transformations and assignments for the response and the fault message. You can forward the response and the fault message to another service or event. Otherwise, you can send them back to the initial caller, if the initial caller is expecting responses and faults.

In case of asynchronous interaction, you can specify a timeout period for receiving the response. The timeout period can be specified in seconds, hours, days, months, or years. By default, the timeout period is infinite. If a callback response does not come within the specified timeout period, then a timeout response can be forwarded to another service, event, or back to the initial caller.

You cannot route a Mediator response to a two-way service. If you want to route a response to a two-way service, then you should use a one-way Mediator in between the first Mediator and the two-way service. The response should first be forwarded to the one-way Mediator, which in turn should call the two-way service.

Note:

- Zero is an unsupported value to be specified as a timeout period.
 - If the callback is received, but processing of callback fails, then by default, the timeout handler is invoked for processing the action specified in the timeout handler.
 - Typically, the caller receives the callback after waiting for 100 milliseconds. But, if you have a bridge Mediator with a sequential routing rule and a connection to a synchronous interface service, then due to the complex flow of the program with all sequential routing rules, the caller may take longer time to be ready to receive the callback. You can work around this issue by changing the routing rule of the bridge Mediator to parallel.
-
-

Specifying a Timeout Period

Perform the following steps to specify a timeout period:

1. Click the **Browse for target service operation** icon next to the <<**Target Operation**>> field in the Callback section.

The Target Type dialog is displayed.

2. Select Service or Event.

The Target Service or the Event Chooser dialog is displayed depending upon the selection you made.

3. Select an event or service.
4. Click **OK**.

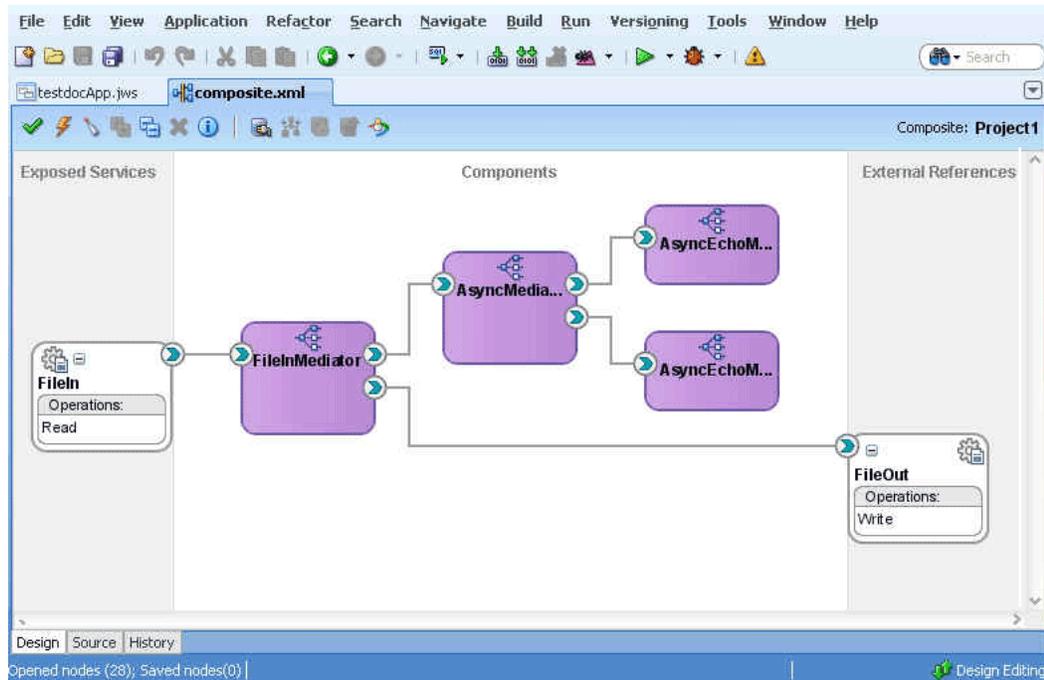
The timeout response will be forwarded to the specified service or event.

Note: If the number of routing rules is larger, and the time taken to execute the routing rules exceeds the transaction timeout, then you must set the transaction timeout to a value that is greater than the time taken to execute all the routing rules.

19.2.2.4 Handling Multiple Callbacks

A single Mediator cannot handle multiple callbacks. If you have a composite application with a Mediator that receives multiple callbacks, then the behavior of the composite application is undetermined. For example, consider the scenario shown in [Figure 19–9](#), where, `AsyncMediator` forwards the callback response from `AsyncEchoMediator1` and `AsyncEchoMediator2` to `FileInMediator`. In such a flow, the `AsyncMediator` may return the callback from both the `AsyncEchoMediator1` and `AsyncEchoMediator2`, or from either one of them. The exact behavior is stochastic and cannot be predicted.

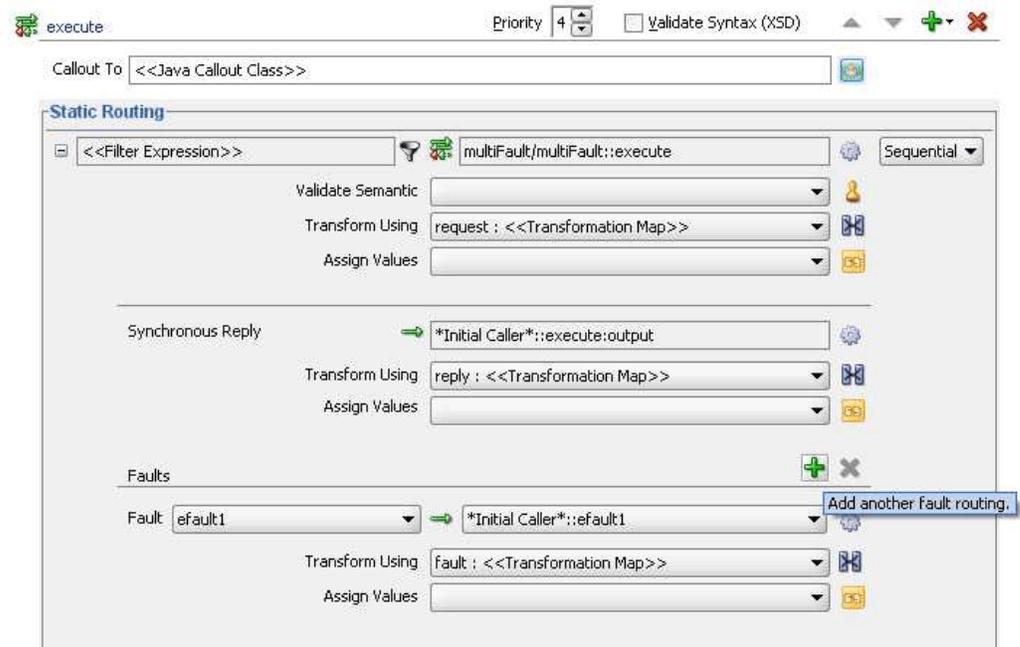
Figure 19–9 Sample Mediator Handling Multiple Callback



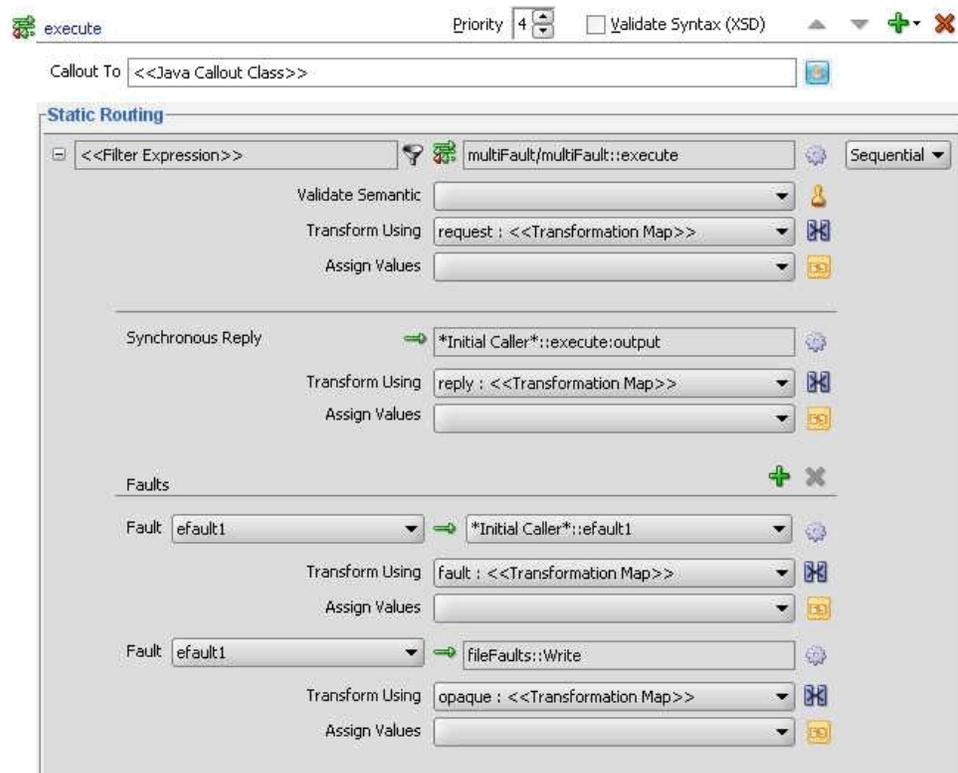
19.2.2.5 Handling Faults

If you create a new routing rule, where the target service WSDL operation has one or more faults, then you will still see a single Fault routing section in the Mediator Editor window. If the source Mediator service supports one or more faults, then the fault is routed back to the caller by default. You can choose the source and target fault names to be routed. You may also use the service browser to route the fault to another target.

To route another fault, click **Add another fault routing** button shown in the following figure:

Figure 19–10 Adding a Second fault

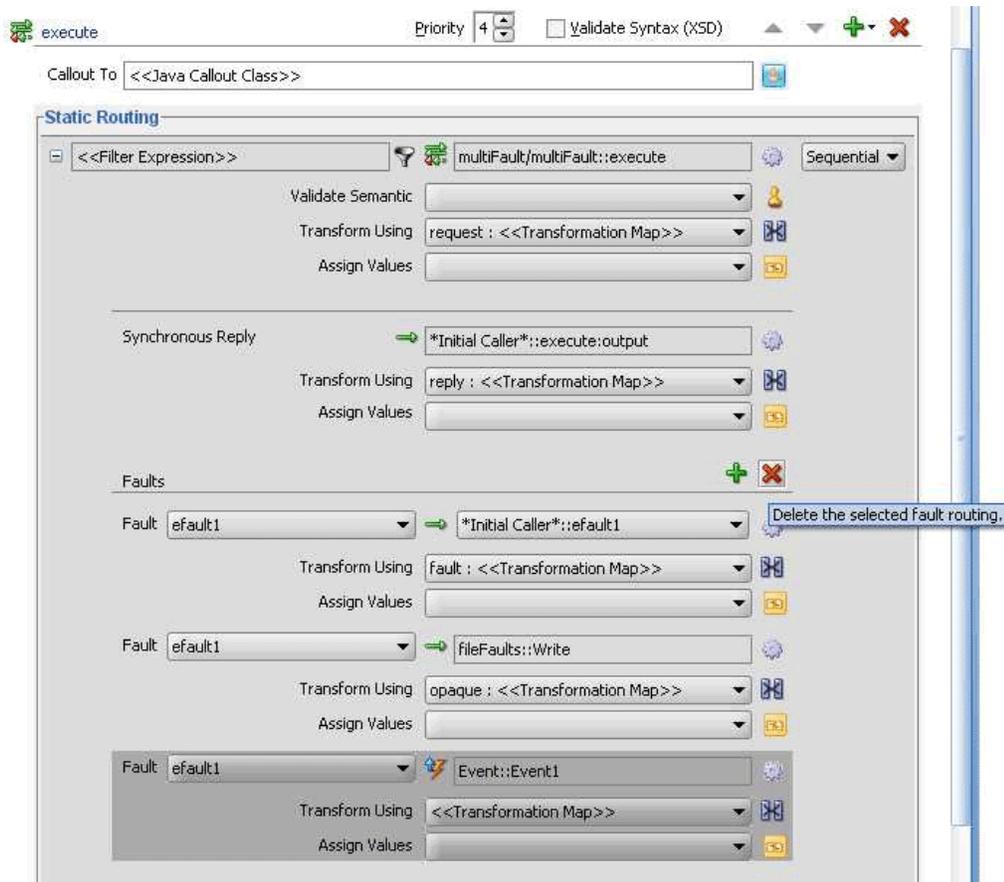
This will add another fault section to the routing rule. In the following figure, a second fault is being routed to a File adapter service:

Figure 19–11 Adding a Second Fault

Note: It is possible to route the same fault to many different targets using different transformations.

While choosing the target for a fault routing, if you want to remove a fault routing section, then you must click **Delete the selected fault routing** as shown in Figure 19–12.

Figure 19–12 Deleting a Fault Routing



Otherwise, you can also click **Clear Target** on the Target Type dialog as shown in the following figure:

Figure 19–13 Target Type Dialog



19.2.2.6 Specifying Expression for Filtering Messages

The filter expression routing rule enables you to filter messages based on their payload. If the filter expression for a given message instance evaluates to true, then the message is delivered to the target service or event specified within the routing rule.

For example, suppose you want to route your data to customers in two different countries: US and Canada. However, you only want notices regarding the product line of type MOBILE to be sent to the customers in US and the product line of type LANDLINE to the customers in Canada. To implement this routing, you must define a routing rule for each component/operation pair that sends messages to the target customers. In addition, you specify filter expressions for the routing rules that send messages to the customers in US or Canada.

You can also define filter expressions message properties or message headers.

Filter Expression Message Properties

Following two are examples of filter expressions message properties:

```
$in.property.custom.Priority = '1'
```

```
$in.property.tracking.ecid = '2'
```

Filter Expression Message Headers

Following two are examples of filter expressions message headers:

```
$in.header.wsse_Security/wsse:Security/Priority = '234'
```

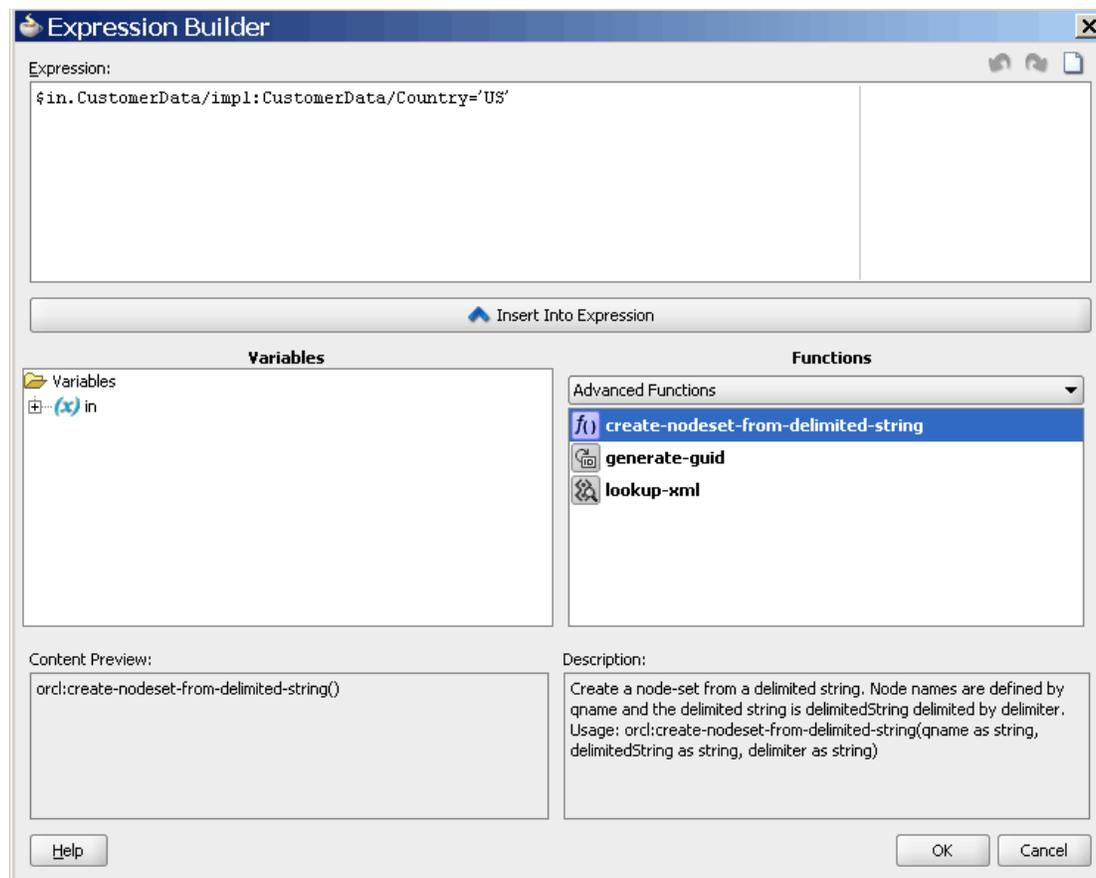
```
$in.header.wsse_Security/wsse:Security/Priority = '234'
```

For the preceding filter expressions message headers to work, you must add the following attribute to the root element of the `.mplan` file:

```
wsse =
"http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd
"
```

You can specify a filter expression by using the Expression Builder dialog as shown in [Figure 19-14](#). The Expression Builder dialog is displayed when you click the icon to the right of the **filter expression** field in the Routing Rules panel.

Figure 19–14 Expression Builder Dialog



The Expression Builder dialog contains the components and controls that assist you in designing a filter expression. Briefly, you double-click a value in the **Variables** field or the Functions palette, to add the value to the **Expression** field. Using a combination of Variable elements, functions, and manually entered text, you can build an expression by which you want message payloads to be filtered for a given routing rule.

The following list describes each of the fields in the Expression Builder dialog:

- **Expression field**

You can enter the filter expression – either manually, or by using the **Variable** field and the Functions palette in this field.

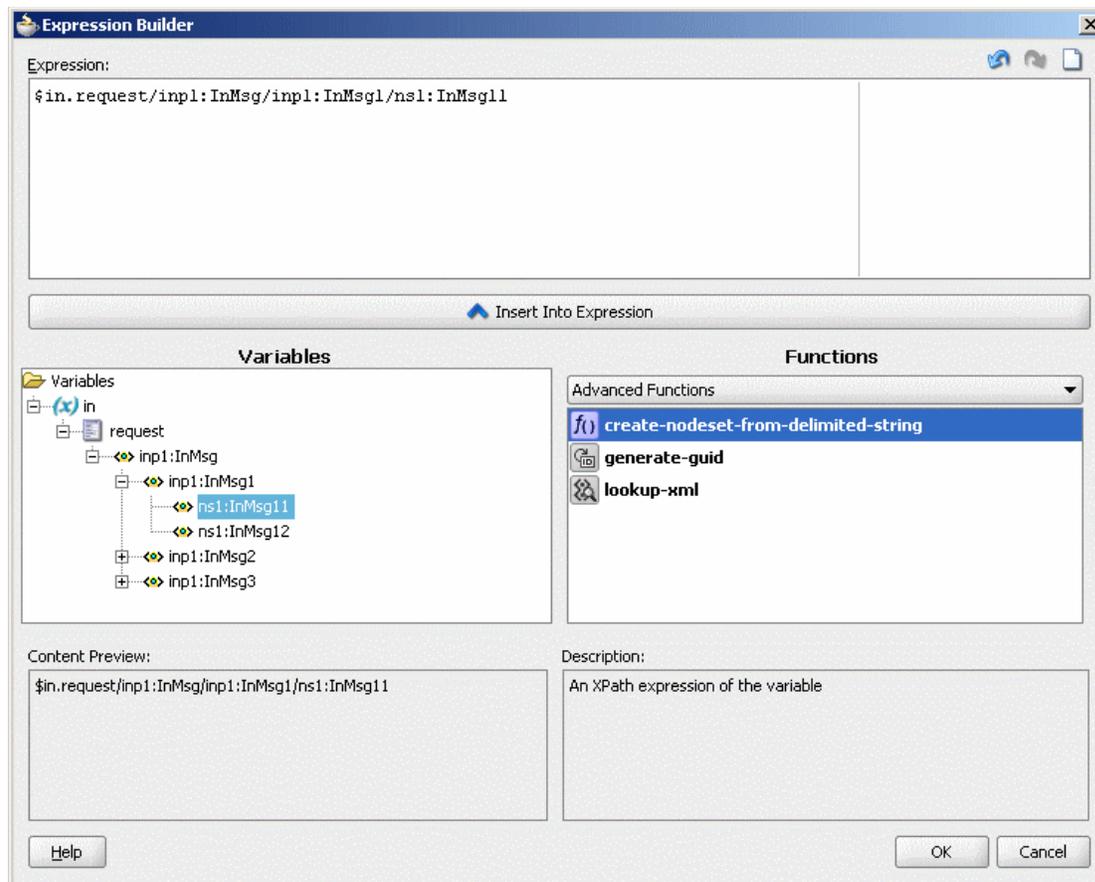
The icons on the upper right side of this field enable you to undo the last edit made, redo the last edit made, or clear the entire **Expression** field, respectively.

- **Variables field**

This field contains the message defined for a Mediator. Oracle JDeveloper parses the Mediator WSDL file and presents the message definition in the **Variables** field. The input message is stored in the `$in` variable. You can use `$in.properties` to access properties of an input message.

If the input message consists of multiple parts, then you can use `$in.<partname>` to access a part of an input message as shown in [Figure 19–15](#).

Figure 19–15 Multiple Part Message in Expression Builder



- Functions Palette

This list enables you to select different functions to include in an expression. When you select a function, a preview of how that function will appear when added to the **Expression** field is presented in the **Content Preview** field, and a description of the function is presented in the **Description** field.

- Content Preview

This field indicates how a value selected from the **Variables** field or Functions palette will appear when it is inserted into the **Expression** field.

- Description

This field describes the value selected from the **Variables** field or Functions palette.

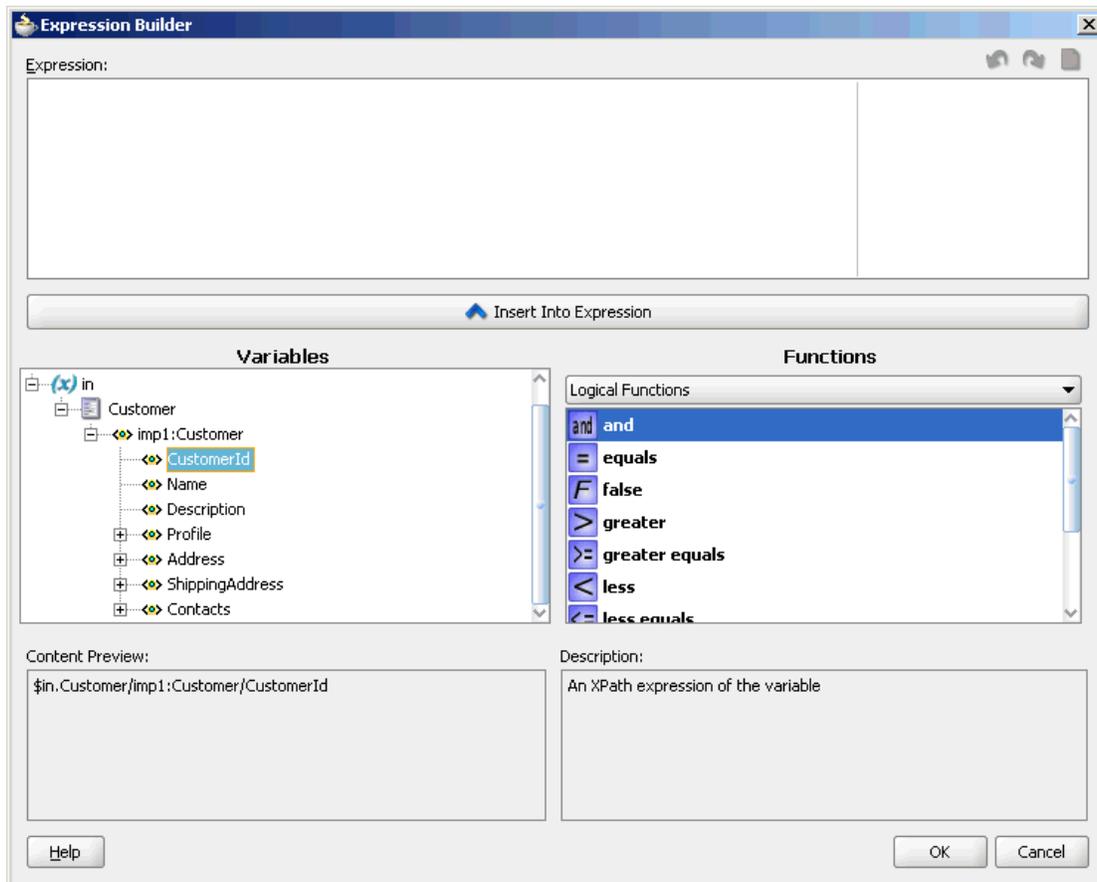
To specify a filter expression on a message payload, follow these steps:

1. In the Routing Rules panel, click the **Add Filter Expression** icon, shown in [Figure 19–2](#).

The Expression Builder dialog is displayed.

2. In the **Variables** field, expand the message definition and select the message element on which you want to base the expression. For example, `CustomerID` element is shown selected in [Figure 19–16](#).

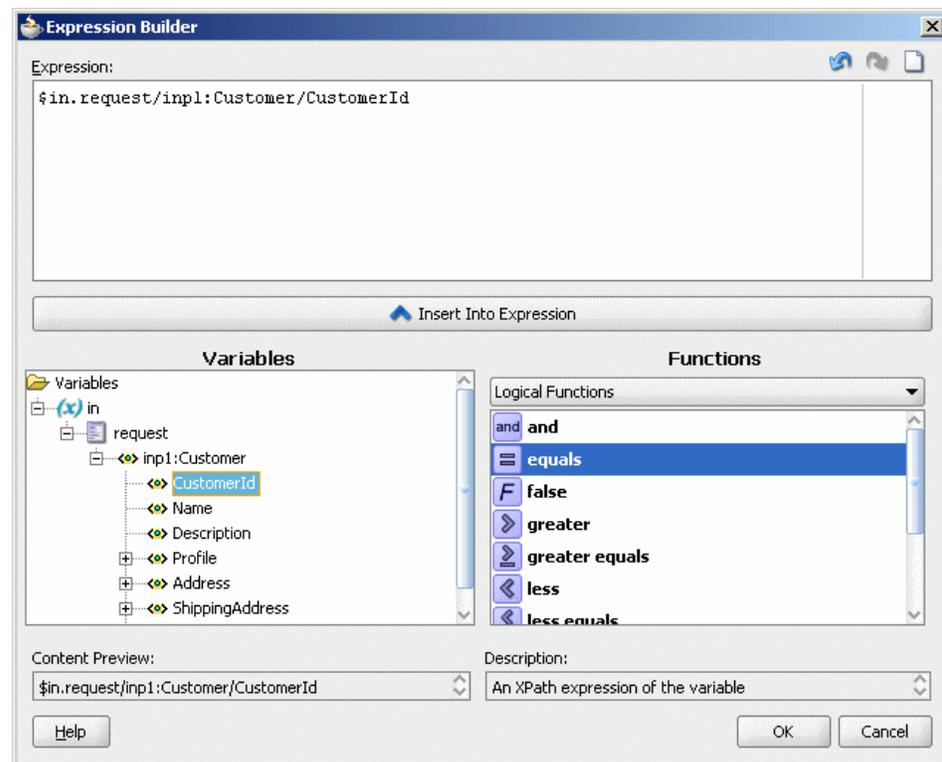
Figure 19–16 Expression Builder Dialog – Variables Element Selected



3. Click **Insert Into Expression**.

The expression is added in the **Expression** field, as shown in [Figure 19–17](#).

Figure 19–17 Expression Builder Dialog – Variables Element Inserted



4. From the **Function** list, select the function to apply to the message payload. For example, `equals`.

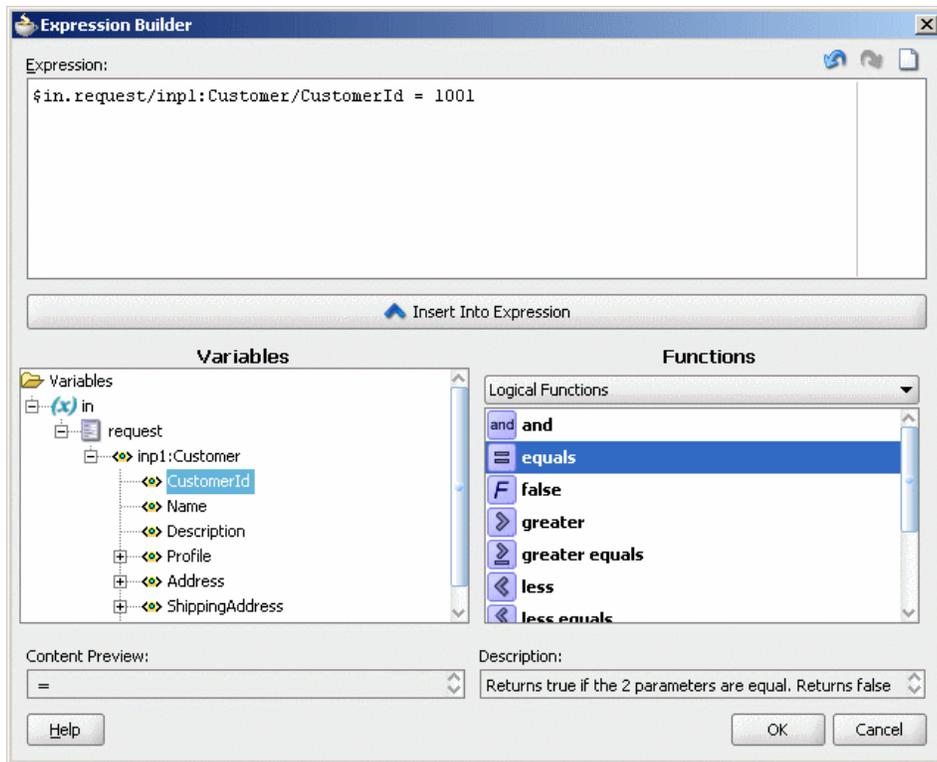
Functions are grouped in categories that are listed when you click the down arrow in the Functions list. For example, if you click the down arrow and select Logical Functions, the list appears as shown in Figure 19–17. When you select a function within the Logical Functions list, a description of that function is presented in the Description box.

5. Click **Insert Into Expression**.

The XPath expression for the selected function is inserted in to the **Expression** field.

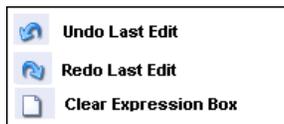
6. Complete the expression. In this example, a value of 1001 is entered, as shown in Figure 19–18.

Figure 19–18 Sample Expression Builder Dialog – Value Entered



7. You can edit the expression manually, or use the expression editing icons, which are summarized in [Figure 19–19](#).

Figure 19–19 Expression Editing Icons



8. Click **OK**.

The expression is added to the Routing Rule panel.

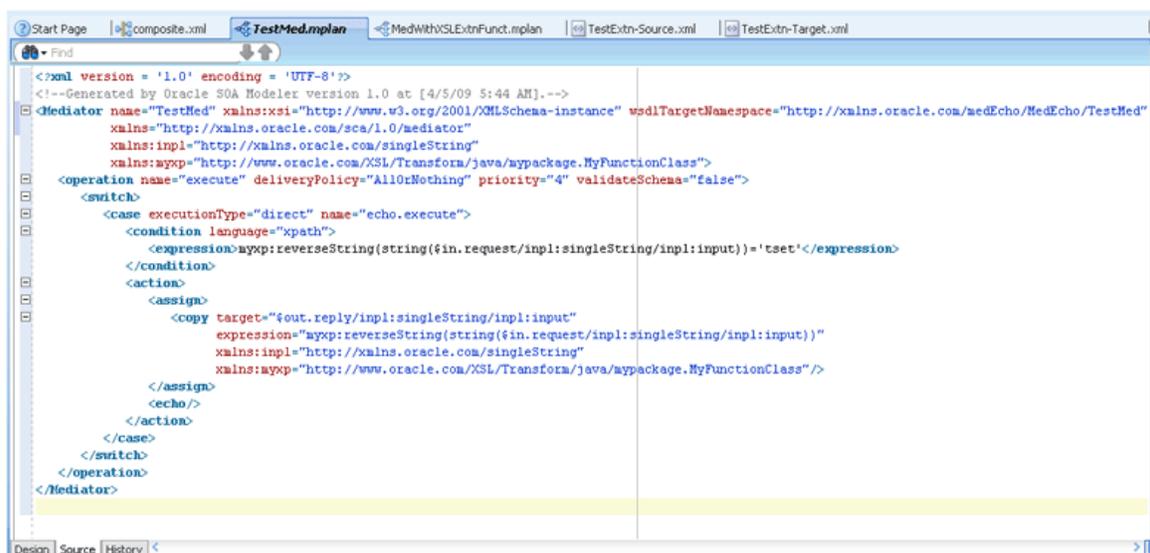
To modify or delete a filter expression, double-click the **Add Filter Expression** icon, and then modify or delete the expression in the **Expression** field of the Expression Builder.

19.2.2.6.1 Using User-Defined Extension Functions You can use the Expression Builder to use the user-defined extension functions. Perform the following steps to use the user-defined extension functions:

1. Create an XPath function.
2. Register the Jaxen XPath function with a Mediator component in the `xpath-function.xml` file on the server side.
3. Open JDeveloper.
4. Use the Builder Expression to customize the expression.
5. Deploy the JDeveloper project to WLS.

6. Copy the JAR file containing the user-defined extension functions to the `beahome/user_projects/domains/soainfra/autodeploy/soa-infra/APP-INF/lib` directory.
7. Modify the `.mplan` file of the project in the following way:
 - Add the function namespace you have defined for the extension functions under Mediator element
 - Add the function names under Expression element
 This has been illustrated in [Figure 19–20](#).

Figure 19–20 Project `.mplan` file – Modified to Use User-Defined Extension Functions



```

<?xml version = '1.0' encoding = 'UTF-8'?>
<!--Generated by Oracle SOA Modeler version 1.0 at [4/5/09 5:44 AM].-->
<Mediator name="TestMed" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" wsdlTargetNamespace="http://xmlns.oracle.com/medEcho/MedEcho/TestMed"
xmlns="http://xmlns.oracle.com/sca/1.0/mediator"
xmlns:inpl="http://xmlns.oracle.com/singleString"
xmlns:myxp="http://www.oracle.com/XSL/Transform/java/mypackage.MyFunctionClass">
  <operation name="execute" deliveryPolicy="AllOrNothing" priority="4" validateSchema="false">
    <switch>
      <case executionType="direct" name="echo.execute">
        <condition language="xpath">
          <expression>myxp:reverseString(string($in.request/inpl:singleString/inpl:input))='tset'</expression>
        </condition>
        <action>
          <assign>
            <copy target="$out.reply/inpl:singleString/inpl:input"
              expression="myxp:reverseString(string($in.request/inpl:singleString/inpl:input))"
              xmlns:inpl="http://xmlns.oracle.com/singleString"
              xmlns:myxp="http://www.oracle.com/XSL/Transform/java/mypackage.MyFunctionClass"/>
          </assign>
          <echo />
        </action>
      </case>
    </switch>
  </operation>
</Mediator>

```

8. Invoke the test page with a suitable payload.

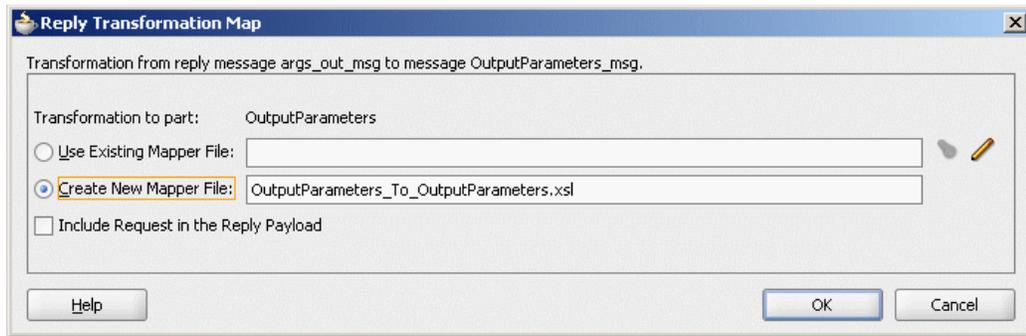
19.2.2.7 Creating Transformations

Oracle JDeveloper provides an XSLT Data Mapper tool that enables you to specify a mapper file (XSL file) to transform data from one XML schema (expressed as an XSD file) to another. This tool 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.

When you click the **transformation map** icon to the right of the **Transform Using** field in the Routing Rules panel, the Request Transformation Map dialog is displayed. You can select an existing XSL file or create a new XSL file with the Data Mapper tool to perform the required transformation.

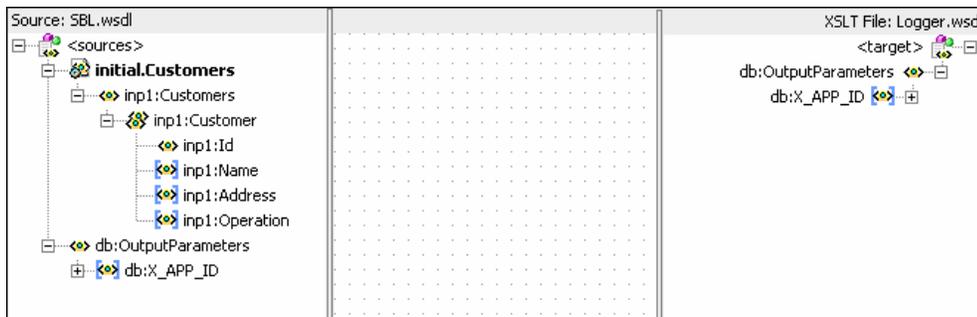
You can also specify transformations for a synchronous reply, callback response message, or a fault message. In case of synchronous reply or fault message, the Reply Transformation Map dialog or the Fault Transformation Map dialog contains the Include Request in the Reply Payload option. [Figure 19–21](#) shows a Reply Transformation Map dialog with this option.

Figure 19–21 Reply Transformation Map Dialog



When you select this option, an `$initial` variable is created which contains the original message of a synchronous interaction as shown in [Figure 19–22](#).

Figure 19–22 Initial Variable in XSL File



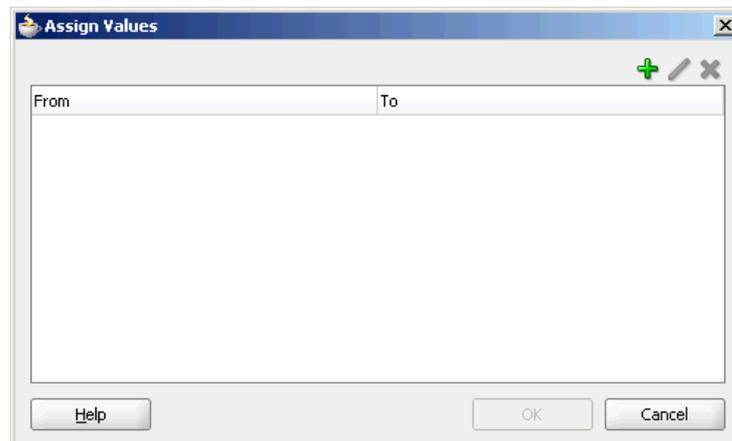
An initial message can also consist of multiple parts. You can use `$initial.<partname>` to access a part of the initial message.

Note: If the parts of the inbound and outbound messages are identical, then no transformation is required for data interchange.

For information about the Data Mapper tool, see [Chapter 45, "Creating Transformations with the XSLT Mapper"](#).

19.2.2.8 Assigning Values

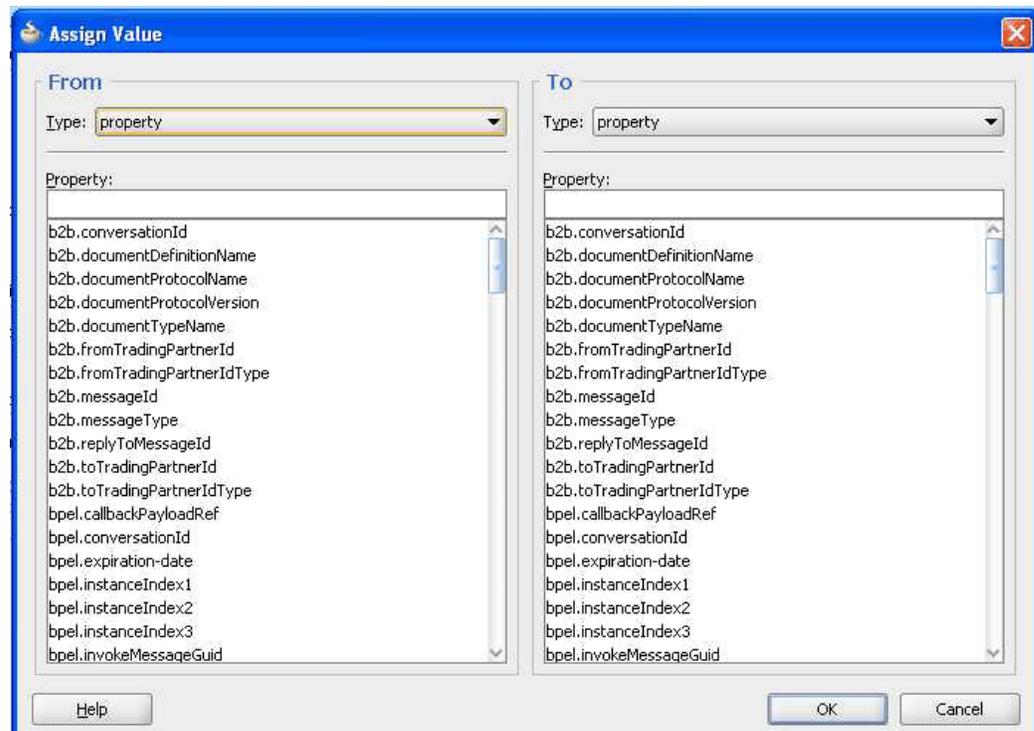
You can use the **Assign Values** field to specify the properties of a target message. [Figure 19–23](#) shows the Assign Values dialog that is displayed when you click the **Assign Values** icon in the routing rules panel.

Figure 19–23 Assign Values Dialog

To set the properties of the target message:

1. Click **Add** in the Assign Values dialog.

The Assign Value dialog is displayed as shown in [Figure 19–24](#).

Figure 19–24 Assign Value Dialog

2. In the From section, select any of the following options from Type box:
 - **Property:** Select this option to assign value of a property to the target message. The property list contains a list of predefined message properties. You can also enter any user-defined property name.
 - **Expression:** Select this option to assign value of an expression to the target message. When you click the **Invoke Expression Builder** icon to the right of

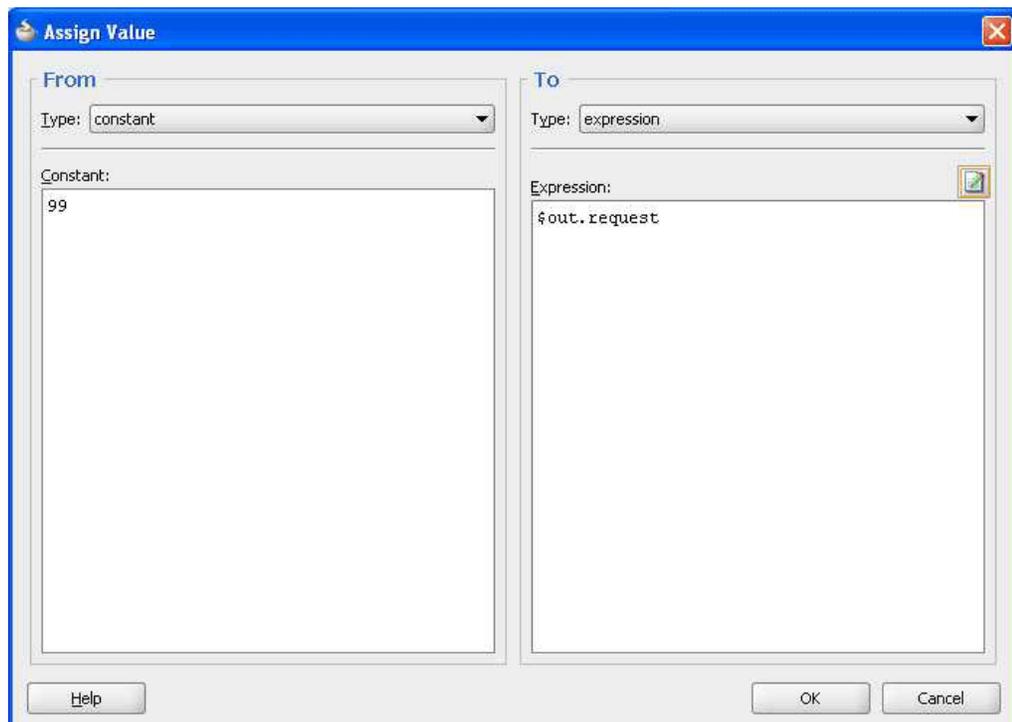
Expression field, the Expression Builder dialog similar to the one shown in [Figure 19–14](#) is displayed.

For more information about the Expression Builder dialog, see [Section 19.2.2.6, "Specifying Expression for Filtering Messages"](#).

- Constant: Select this option to assign a constant value to the target message.
3. In the To section, select any of the following options:
- Property: Select this option to copy the value to a message property. The **Variable** field of the Expression Builder dialog contains an `$out` variable that contains the output message. You can use `$out.properties` to access properties of an output message.
 - Expression: Select this option to copy the value to an expression. When you click the **Invoke Expression Builder** icon to the right of **Expression** field, the Expression Builder dialog is displayed. The **Variable** field of the Expression Builder dialog contains an `$out` variable that contains the output message. You can use `$out.<partname>` to access a complete output message or part of an output message. Note that you cannot assign any value after the `<partname>`. For example, in [Figure 19–25](#), the expression is `$out.request` and you cannot modify it to add any value after `request`.

[Figure 19–25](#) shows a sample Assign Value dialog in which a constant value is specified as an expression.

Figure 19–25 Populated Assign Value Dialog



4. Click **OK** in the Assign Value dialog.
5. Click **OK**. The expression is added to **Assign Values** field of the Routing Rules panel.

Note:

- When you assign values to a particular Mediator property during an event publishing, the assigned value does not get propagated to the subscribing event.

You can work around this issue by using transformations to have the property as part of the event body.

- You cannot assign values to the `jca.db.userName` and `jca.db.password` properties on WebLogic Server because their data sources do not support setting user name or password dynamically to the `getConnection` method.
- By default, SOAP headers are not passed through by Mediator. You must add the `passThroughHeader` endpoint property to the corresponding Mediator routing service. For adding this property, modify the `Composite.xml` file in the following way:

```
<component name="Mediator1">
  <implementation.mediator src="Mediator1.mplan"/>
  <property name="passThroughHeader">true</property>
</component>
```

19.2.2.9 Access Headers for Filters and Assignments

When the Expression Builder is invoked from a Mediator, either for defining a filter or for defining an assignment source or target, the WSDL file is parsed. This automatically detects any SOAP headers for the current routing rule operation and makes them visible as Variables under the `in` or `out` folder as `header./ns_elementName/`, as shown in [Figure 19-26](#). Here, `ns` is the namespace prefix and `elementName` is the root element name for the header schema.

For example:

Example 1

Suppose, the namespace prefixes `wsse` and `ns1` are already defined in the WSDL file or the `.mplan` file, then you can write an XPath expression as the following:

```
$in.header.wsse_Security/wsse:Security/ns1:Foo/Priority
```

Example 2

Suppose, you want to use a schema that does not have a namespace predefined in the WSDL file, then the Expression Builder is enhanced to allow you to enter `{full_namespace}` instead of a prefix. The Expression Builder then generates a unique prefix and the prefix definition is added to the `.mplan` file.

For example, enter the following expression in the Expression Builder:

```
$in.header.{http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd}_Security/
{"http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd"}:
Security/{"http://www.globalcompany.com/ns/OrderBooking"}:Foo/Priority
```

The `.mplan` file will contain the following:

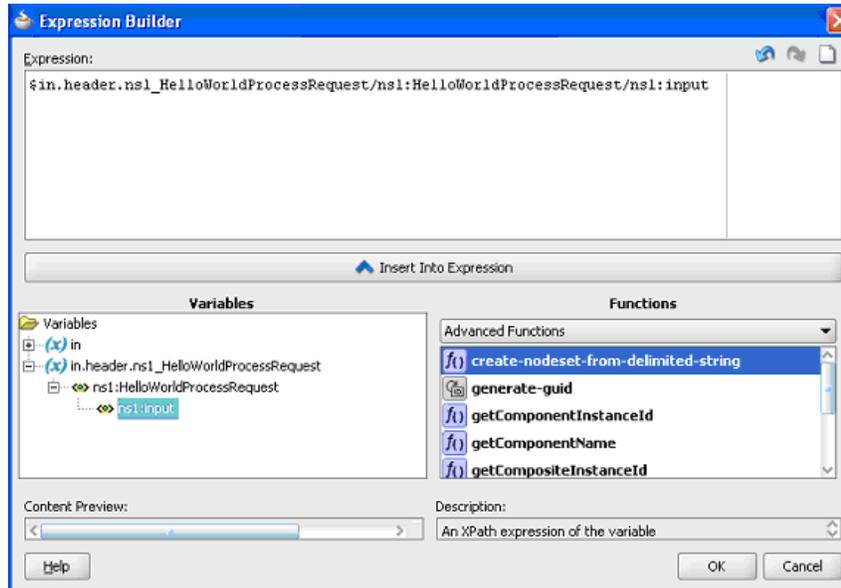
```
xmlns:ns1="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-sece
```

```

xt-1.0.xsd"
xmlns:ns2="http://www.globalcompany.com/ns/OrderBooking"
...
expression="$in.header.ns1_Security/ns1:Security/ns2:Foo/Priority"

```

Figure 19–26 Expression Builder Dialog - Automatic Header Detection



By default, SOAP headers are not passed through by Mediator. You must add the following endpoint property to the corresponding Mediator routing service:

```
<property name="passThroughHeader">true</property>
```

For example, to add this property, you can modify the `Composite.xml` file in the following way:

```

<component name="Mediator1">
  <implementation.mediator src="Mediator1.mplan"/>
  <property name="passThroughHeader">true</property>
</component>

```

Note:

- The UI supports both SOAP 1.1 and SOAP 1.2.
 - For automatic header detection, a concrete WSDL file must be used, when creating the Mediator component.
 - Assignments execute after filters. So, if you are assigning a value in a custom header, then the particular assignment will not be visible to the filter.
-

19.2.2.9.1 Manual Expression Building for accessing Headers for Filters and Assignments

There are use cases, where the header schemas cannot be determined from the WSDL files. For example, security headers that are appended to message, or the headers for a Mediator that was created using an abstract WSDL file. To access these headers, you must manually type in the XPath into the Expression Builder.

The syntax for header expressions is:

```
$in.header.<header root element namespace prefix>_<header root element name>/<xpath>
```

So, for the following header:

```
<wsse:Security
xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-sec
ext-1.0.xsd">
<Priority>234</Priority>
</wsse:Security>
```

The filter expression will be:

```
$in.header.wsse_Security/wsse:Security/Priority = '234'
```

The assignment expression will be:

```
<copy target="$out.property.jca.jms.priority"
expression="$in.header.wsse_Security/wsse:Security/Priority"/>
```

For the preceding expressions to work, you must add the following attribute to the root element of the `.mpplan` file:

```
wsse =
"http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd"
"
```

19.2.2.9.2 Manual Expression Building for Accessing Properties for Filters and Assignments

Example of a filter expression is

```
$in.property.tracking.ecid = '2'
```

Example of an assignment expression is

```
<copy target="$out.property.tracking.ecid" value="$in.property.tracking.ecid"/>
```

19.2.2.10 Using Semantic Validation

You can specify Schematron files for validating an inbound message and its various parts. Schematron version 1.5 is the supported version.

Perform the following steps for specifying a Schematron schema to validate an inbound message and its various parts:

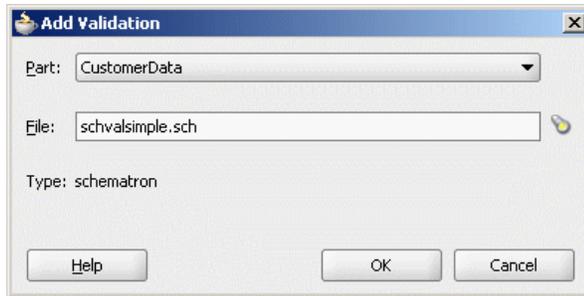
1. Click the **Select Validation File** icon to the right of **Validate Semantic** field.
The Validations dialog is displayed.
2. Click **Add**.
The Add Validation dialog is displayed.
3. From the **Part** list, select a message part.
4. Click **Search** to the right of the **File** field.
The SCA Resource Browser dialog is displayed.
5. Select a Schematron file and click **OK**.

Note:

- Schematron files usually have a .sch extension.
- No error message or warning is displayed if the selected Schematron file is empty.

The Add Validation dialog is updated, as shown in [Figure 19–27](#).

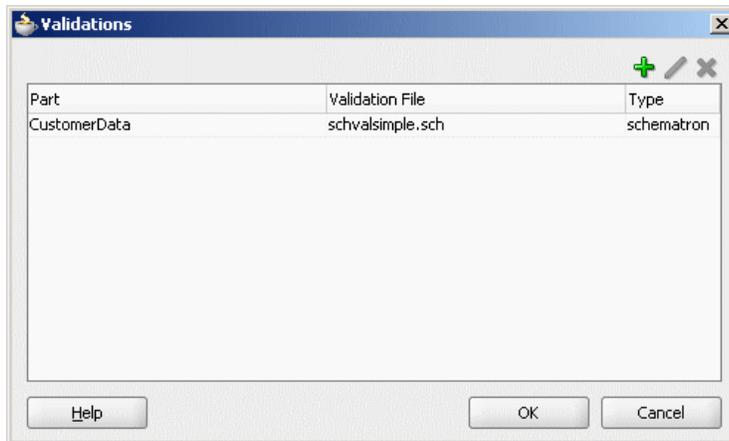
Figure 19–27 Add Validation Dialog



6. Click **OK**.

The Validation dialog is updated, as shown in [Figure 19–28](#).

Figure 19–28 Validation Dialog



7. Click **Add** to specify a Schematron file for another message part or click **OK**.

For more information about building a Schematron schema, refer to the resources available at

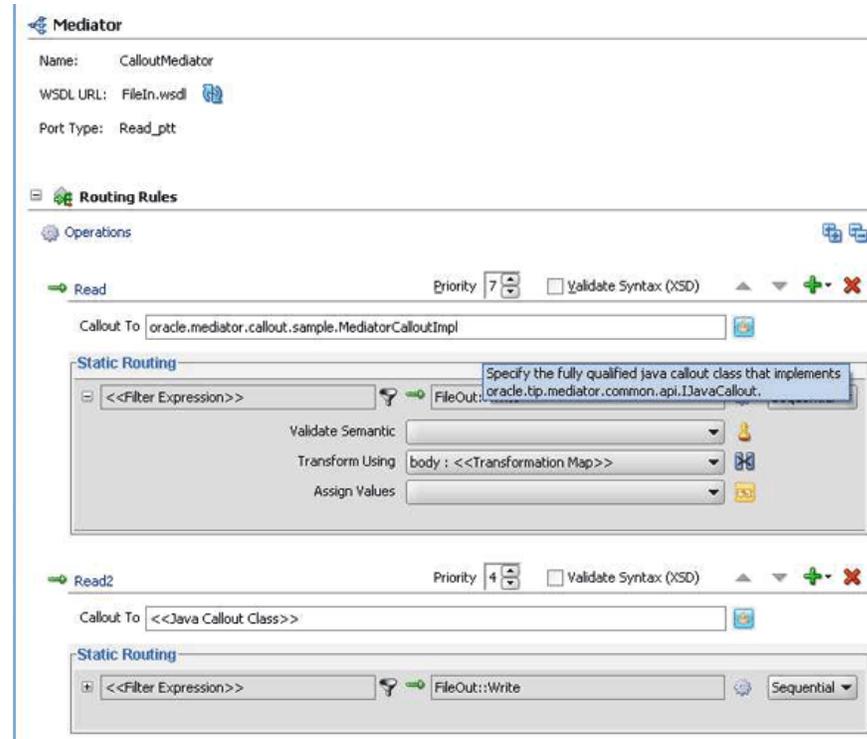
<http://www.schematron.com>

Note: In semantic validation, if you check for the length of each element name, then the element name may change for different set of input. This happens when there are white spaces between nodes because the parser treats the white spaces as test nodes.

19.2.2.11 Support for Java Callouts

Java callouts enable you to use external Java classes to manipulate messages flowing through the Mediator. Only one Java callout is supported per WSDL operation or event subscription. The callout class must implement the `oracle.tip.mediator.common.api.IJavaCallout` interface. Callouts are available for both static and dynamic routings. Figure 19–29 shows a sample Mediator with two operations, where both the operations have one routing rule each and the first operation has a callout class.

Figure 19–29 Sample Mediator Supporting Java Callout



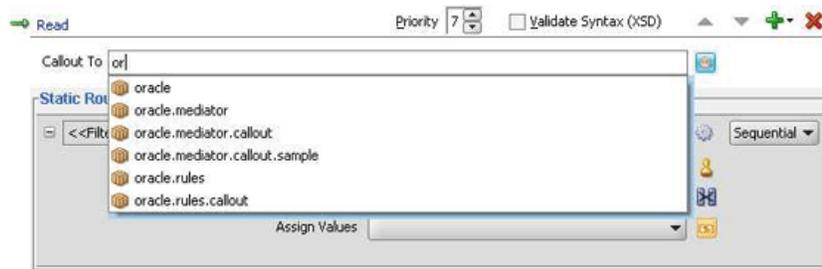
You must ensure that the Java callout class is available on the server. You can use any of the following methods for this:

- Copy the Java class to the SCA-INF/classes folder
- Copy the JAR containing the Java class to the SCA-INF/lib folder
- Copy the JAR containing the Java class to the `$DOMAIN_HOME/lib` folder

If you want to make the Java callout class to be available to multiple Mediators, then you must copy the JAR containing the Java class to the `$DOMAIN_HOME/lib` folder.

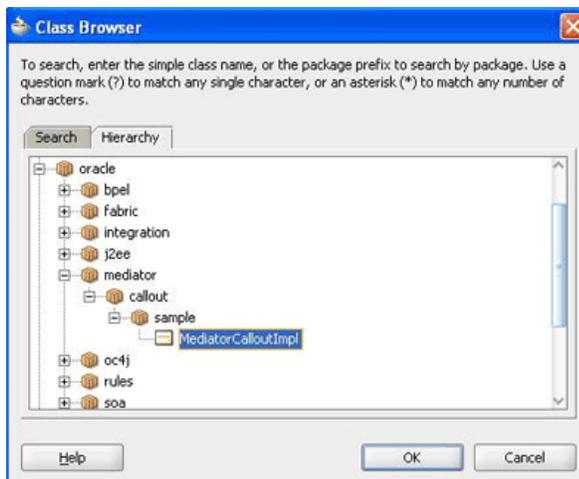
You can manually enter the name of the Java callout class in the **Callout To** field as shown in Figure 19–30. In this case, JDeveloper's auto-completion information feature will complete the address, the classes in the current project.

Figure 19–30 Callout To Field



You can also click **Select java callout class** button to invoke the standard JDeveloper Class Browser as shown in Figure 19–31.

Figure 19–31 JDeveloper Class Browser



The Class Browser is filtered so that it only displays classes that implement the `oracle.tip.mediator.common.api.IjavaCallout` interface.

If you have a Java callout in Mediator and use a filter expression in the same Mediator, then you must set the root element for the payload as follows:

```
changexml doc = XmlUtils.getXmlDocument(ChangedDoc);
String mykey = "request";
message.addPayload(mykey, changexml doc.getDocumentElement());
```

Table 19–1 discusses the methods in the `oracle.tip.mediator.common.api.IjavaCallout` interface.

Table 19–1 Description of Methods in the IjavaCallout Interface

Method	Description
<code>initialize</code>	This method is invoked when the callout implementation class is instantiated for the first time.
<code>preRouting</code>	This method is called before Mediator starts executing the cases. You can customize this method to include validations and enhancements.
<code>preRoutingRule</code>	This method is called before Mediator starts executing any particular case. You can customize this method to include case-specific validations and enhancements.

Table 19–1 (Cont.) Description of Methods in the `ljavaCallout` Interface

Method	Description
<code>preCallbackRouting</code>	This method is called before Mediator finishes executing callback handling. You can customize this method to perform callback auditing and custom fault tracking.
<code>postRouting</code>	This method is called after Mediator finishes executing the cases. You can customize this method to perform response auditing and custom fault tracking.
<code>postRoutingRule</code>	This method is called after Mediator starts executing the cases. You can customize this method to perform response auditing and custom fault tracking.
<code>postCallbackRouting</code>	This method is called after Mediator finishes executing callback handling. You can customize this method to perform callback auditing and custom fault tracking.

Note: If you change the message properties of a Mediator by using Java callout in the `preRoutingRule` method or the `preRouting` method, then you must explicitly copy the changed property to the outbound message by using Mediator assignment functionality. For example, if you are changing the `jca.file.FileName` property in Java callout, then you must update the Mediator assignment statement in the following way:

```
<assign>
<copy target="$out.property.jca.file.FileName"
expression="$in.property.jca.file.FileName"/>
</assign>
```

Table 19–2 discusses the methods in the `CalloutMediatorMessage` interface.

Table 19–2 Description of Methods in the `CalloutMediatorMessage` Interface

Method	Description
<code>addPayload</code>	This method sets payload of the Mediator messages.
<code>addProperty</code>	This method adds property to the Mediator messages.
<code>addHeader</code>	This method adds header to the Mediator messages.
<code>getProperty</code>	This method retrieves Mediator message properties by providing the property name.
<code>getProperties</code>	This method retrieves Mediator message properties.
<code>getId</code>	This method retrieves instance ID of the Mediator messages. This instance ID is the Mediator instance ID created for that particular message.
<code>getPayload</code>	This method retrieves payload of the Mediator messages.
<code>getHeaders</code>	This method retrieves header of the Mediator messages.
<code>getComponentDN</code>	This method retrieves componentDN for the Mediator component.

Note:

- The `oracle.tip.mediator.common.api.AbstractJavaCalloutImpl` class is a dummy implementation¹ of the `IJavaCallout` interface. This class defines all the methods present in the `IJavaCallout` interface. So, you can extend this class to override only a few specific methods of the `IJavaCallout` interface.
- Details of the processing happening within the Java callout, are not displayed in the Mediator Audit Trail screen.

¹ Dummy implementation of an interface means that the implementation class will provide definition for all the methods declared in the particular interface, but one or more defined methods may have an empty method body. Extending a dummy implementation class is much easier because you can choose to override only a subset of the methods, unlike implementing an interface and defining all the methods.

Sample Java Callout Class

The following example shows a sample Java callout class:

```
package qa.as11tests.javacallout;

import com.collaxa.cube.persistence.dto.XmlDocument;

import com.oracle.bpel.client.NormalizedMessage;

import java.util.logging.Logger;
import java.util.Map;
import java.util.Iterator;

import oracle.tip.mediator.common.api.CalloutMediatorMessage;
import oracle.tip.mediator.common.api.ExternalMediatorMessage;
import oracle.tip.mediator.common.api.IJavaCallout;
import oracle.tip.mediator.common.api.MediatorCalloutException;
import oracle.tip.mediator.metadata.CaseType;
import oracle.tip.mediator.utils.XmlUtils;

import oracle.tip.pc.services.functions.ExtFunc;

import oracle.xml.parser.v2.XMLDocument;

import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;

public class JavaCalloutSanity implements IJavaCallout {
    Logger logger = Logger.getLogger("Callout");
    public JavaCalloutSanity() { }

    public void initialize(Logger logger) throws MediatorCalloutException {
        this.logger = logger;
        this.logger.info("Initializing...");
    }
    public boolean preRouting(CalloutMediatorMessage calloutMediatorMessage) {
        System.out.println("Pre routing...");
        String sPayload = "null";
        String sPayload_org = "null";
        for (Iterator msgIt =
calloutMediatorMessage.getPayload().entrySet().iterator());
```

```

        msgIt.hasNext(); ) {
        Map.Entry msgEntry = (Map.Entry)msgIt.next();
        Object msgKey = msgEntry.getKey();
        Object msgValue = msgEntry.getValue();
        if (msgKey.equals("request"))
            sPayload = XmlUtils.convertDomNodeToString((Node)msgValue);
    }
    sPayload_org = sPayload;
    String tobeReplaced = "CHANGE_THIS";
    String replaceWith = "JAVA_CALLOUT_|_|_PRE_ROUTING";
    int start = sPayload.indexOf(tobeReplaced);
    StringBuffer sb = new StringBuffer();
    sb.append(sPayload.substring(0, start));
    sb.append(replaceWith);
    sb.append(sPayload.substring(start + tobeReplaced.length()));
    String changedPayload = sb.toString();
    String uid;
    try {
        uid = ExtFunc.generateGuid();
    } catch (Exception e) {
    }
    XMLDocument changedoc;
    try {
        changedoc = XmlUtils.getXmlDocument(changedPayload);
        String mykey = "request";
        calloutMediatorMessage.addPayload(mykey, changedoc);
        //calloutMediatorMessage.getPayload().put(mykey, changedoc);
    } catch (Exception e) {
    }
    System.out.println("Changed from : \n"+sPayload_
org+"\nTo\n"+changedPayload);
    System.out.println("End Pre routing...\n\n");
    return false;
}
public boolean postRouting(CalloutMediatorMessage calloutMediatorMessage,
        CalloutMediatorMessage calloutMediatorMessage1,
        Throwable throwable) throws
MediatorCalloutException {
    System.out.println("Start Post routing...");
    String sPayload = "null";
    String sPayload_org = "null";
    for (Iterator msgIt =
calloutMediatorMessage1.getPayload().entrySet().iterator();
        msgIt.hasNext(); ) {
        Map.Entry msgEntry = (Map.Entry)msgIt.next();
        Object msgKey = msgEntry.getKey();
        Object msgValue = msgEntry.getValue();
        if(msgKey.equals("reply"))
            sPayload = XmlUtils.convertDomNodeToString((Node)msgValue);
    }

    sPayload_org = sPayload;
    String tobeReplaced = "POST_ROUTING_RULE_REQUEST_REPLY";
    String replaceWith = "POST_ROUTING_RULE_REQUEST_REPLY_|_|_POSTROUTING_|_|_
JAVA_CALLOUT_WORKING";
    int start = sPayload.indexOf(tobeReplaced);
    StringBuffer sb = new StringBuffer();
    sb.append(sPayload.substring(0, start));
    sb.append(replaceWith);
    sb.append(sPayload.substring(start + tobeReplaced.length()));

```

```

        String changedPayload = sb.toString();
        XMLDocument changedoc;
        try {
            changedoc = XmlUtils.getXmlDocument(changedPayload);
            String mykey = "reply";

calloutMediatorMessage1.addPayload(mykey, changedoc.getDocumentElement());
            // calloutMediatorMessage1.getPayload().put(mykey,
changedoc.getDocumentElement());
        } catch (Exception f) {
        }
        System.out.println("Changed from : \n"+sPayload_org+"\nTo\n"+
            changedPayload);
        System.out.println("End Post routing...\n\n");
        return false;
    }
    public boolean preRoutingRule(CaseType caseType,
        CalloutMediatorMessage calloutMediatorMessage) {
        System.out.println("\nStart PreRoutingRule.\n");
        String sPayload = "null";
        String sPayload_org = "null";
        for (Iterator msgIt =
            calloutMediatorMessage.getPayload().entrySet().iterator();
            msgIt.hasNext(); ) {

            Map.Entry msgEntry = (Map.Entry)msgIt.next();
            Object msgKey = msgEntry.getKey();
            Object msgValue = msgEntry.getValue();
            if(msgKey.equals("request"))
                sPayload = XmlUtils.convertDomNodeToString((Node)msgValue);
        }
        sPayload_org = sPayload;
        String tobeReplaced = "PRE_ROUTING";
        String replaceWith = "PRE_ROUTING_||_PRE_ROUTING_RULE";
        int start = sPayload.indexOf(tobeReplaced);
        StringBuffer sb = new StringBuffer();
        sb.append(sPayload.substring(0, start));
        sb.append(replaceWith);
        sb.append(sPayload.substring(start + tobeReplaced.length()));
        String changedPayload = sb.toString();
        XMLDocument changedoc;
        try {
            changedoc = XmlUtils.getXmlDocument(changedPayload);
            String mykey = "request";
            calloutMediatorMessage.addPayload(mykey, changedoc);
            // calloutMediatorMessage.getPayload().put(mykey, changedoc);
        } catch (Exception e) {
        }
        System.out.println("Changed from : \n"+sPayload_
org+"\nTo\n"+changedPayload);
        System.out.println("End PreRoutingRule.\n\n");
        return true;
    }
    public boolean postRoutingRule(CaseType caseType,
        CalloutMediatorMessage calloutMediatorMessage,
        CalloutMediatorMessage calloutMediatorMessage1,
        Throwable throwable) {
        System.out.println("Start PostRoutingRule.");
        String req_sPayload = "null";
        String req_sPayload_org = "null";

```

```

String rep_sPayload = "null";
String rep_sPayload_org = "null";
for (Iterator msgIt =
    calloutMediatorMessage.getPayload().entrySet().iterator();
    msgIt.hasNext(); ) {
    Map.Entry msgEntry = (Map.Entry)msgIt.next();
    Object msgKey = msgEntry.getKey();
    Object msgValue = msgEntry.getValue();
    if(msgKey.equals("request"))
        req_sPayload = XmlUtils.convertDomNodeToString((Node)msgValue);
}
req_sPayload_org = req_sPayload;
String tobeReplaced = "PRE_ROUTING_RULE";
String replaceWith = "PRE_ROUTING_RULE_|_|_POST_ROUTING_RULE_REQUEST";
int start = req_sPayload.indexOf(tobeReplaced);
StringBuffer sb = new StringBuffer();
sb.append(req_sPayload.substring(0, start));
sb.append(replaceWith);
sb.append(req_sPayload.substring(start + tobeReplaced.length()));
String changedPayload = sb.toString();
XMLDocument changedoc;
try {
    changedoc = XmlUtils.getXmlDocument(changedPayload);
    String mykey = "request";
    calloutMediatorMessage.addPayload(mykey, changedoc);
    // calloutMediatorMessage.getPayload().put(mykey, changedoc);
} catch (Exception e) {
}
for (Iterator msgIt =
    calloutMediatorMessage1.getPayload().entrySet().iterator();
    msgIt.hasNext(); ) {
    Map.Entry msgEntry = (Map.Entry)msgIt.next();
    Object msgKey = msgEntry.getKey();
    Object msgValue = msgEntry.getValue();
    if(msgKey.equals("reply"))
        rep_sPayload = XmlUtils.convertDomNodeToString((Node)msgValue);
}
rep_sPayload_org = rep_sPayload;
tobeReplaced = "PRE_ROUTING_RULE";
replaceWith = "PRE_ROUTING_RULE_|_|_POST_ROUTING_RULE_REQUEST_REPLY";
start = rep_sPayload.indexOf(tobeReplaced);
sb = new StringBuffer();
sb.append(rep_sPayload.substring(0, start));
sb.append(replaceWith);
sb.append(rep_sPayload.substring(start + tobeReplaced.length()));
changedPayload = sb.toString();
try {
    changedoc = XmlUtils.getXmlDocument(changedPayload);
    String mykey = "reply";

calloutMediatorMessage1.addPayload(mykey, changedoc.getDocumentElement());
    // calloutMediatorMessage1.getPayload().put(mykey,
changedoc.getDocumentElement());
} catch (Exception e) {
}
    System.out.println("Changed from : \n"+req_sPayload_
org+"\nTo\n"+changedPayload);
    System.out.println("End postRoutingRule\n\n");
    return true;
}
}

```

}

19.2.3 Creating Dynamic Routing Rules

The basic idea behind dynamic routing is to separate the control logic, which determines the path taken by the process, from the execution of the process. In the dynamic routing scenario, a decision matrix is used to determine the type of Level-2 service to be chosen for each routing. The factors that affect the decision on the type of Level-2 service are channel, customer type, and so on. The solution allows this decision matrix to be modified externally by business analysts without changing the routing. The decision matrix must be evaluated to determine the outbound service.

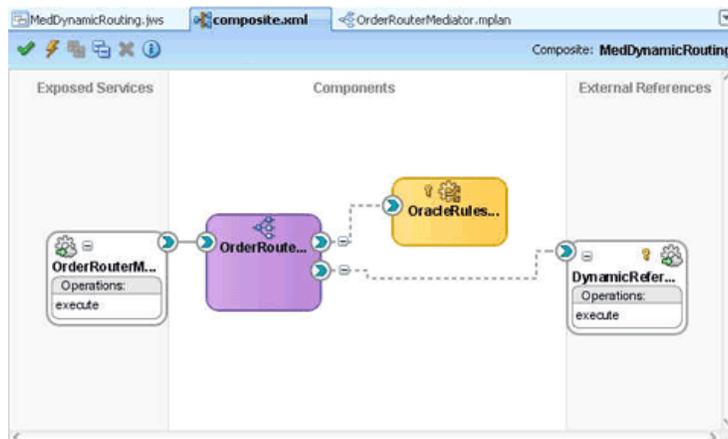
Dynamic routing rules can be created by using the Dynamic Routing Rule option of Mediator Editor window, as shown in Figure 19–32:

Figure 19–32 Mediator Editor Window Displaying Dynamic Routing Rule Option



This creates a new Business Rules component and the Business Rule component is wired to the Mediator component within the SCA composite of the Mediator component. The wire links between the Business Rule component and the Mediator component are considered implementation details and are shown as dotted lines in the SCA editor, as shown in Figure 19–33:

Figure 19–33 SCA Editor with Wire Links Between the Business Rule Component and the Mediator Component



The Business Rule component comprises of a rule dictionary. The rule dictionary is a metadata container for the rule engine artifacts, such as fact types, rulesets, rules,

decision tables and so on. As part of creating the Business Rules component, the rule dictionary is pre-initialized with the following data:

- Fact Type Model

The fact type model is the data model that can be used for modeling rules. The rule dictionary is populated with a fact type model that corresponds to the input of the phase activity, and some fixed data model that is required as part of the contract between the Mediator component and the Business Rules component.

- Ruleset

A ruleset is a container of rules and used as a kind of grouping mechanism for rules. A ruleset can be exposed as a service. As part of creating the Business Rules component, one ruleset is created within the rule dictionary.

- Decision Table (or Matrix)

From a rule engine perspective, a decision table is a collection of rules with the same fact type model elements in the condition and action part of the rules. The decision table enables to visualize rules in a tabular format. As part of creating the Business Rules component, a new decision table is created within the ruleset.

- Decision Service

As part of creating the Business Rules component, a decision service is created to expose the ruleset as a service of the Business Rules SCA component. The service interface is used by the Mediator component to evaluate the decision table.

After all the required artifacts of the phase activity are created, the wizard starts modeling the phase decision matrix (PDM). The wizard launches the Rule Designer window of JDeveloper and enables you to edit the phase decision matrix. Figure 19–34 shows a sample decision table within the Rule Designer:

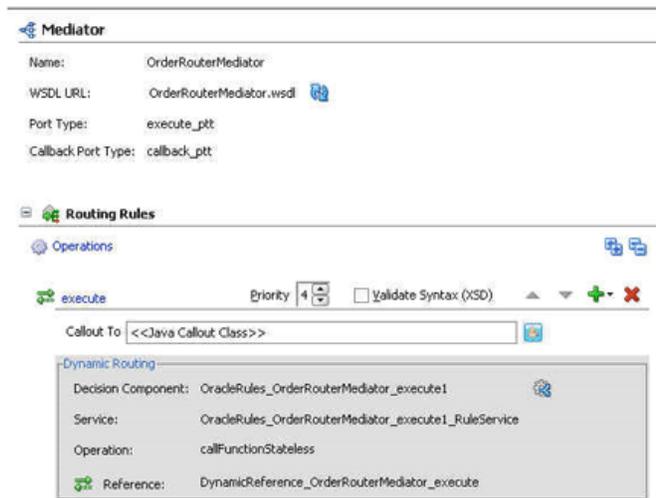
Figure 19–34 Sample Decision Table Within the Rule Designer

Conditions	R1	R2	R3	R4
C1 CustomerData.Type.type	otherwise	"GOLD"	"Silver"	"Platinum"
Actions	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
A1 assert new DynamicRo...	"dynamic_case"	"dynamic_case"	"dynamic_case"	"dynamic_case"
caseName:String	"callback"	"callback"	"callback"	"callback"
cbkOperation:String	"direct"	"direct"	"direct"	"direct"
executionType:String	"callback"	"callback"	"callback"	"callback"
onCbkOperation:String	"http://localhost:8001/defaultService"	"http://localhost:8001/GoldService"	"http://localhost:8001/SilverService"	"http://localhost:8001/PlatinumService"
serviceBindingInfo:String	"execute"	"execute"	"execute"	"execute"
serviceOperation:String	"DynamicReference_OrderRouterMediator_execute"	"DynamicReference_OrderRouterMediator_execute"	"DynamicReference_OrderRouterMediator_execute"	"DynamicReference_OrderRouterMediator_execute"
serviceReference:String				

Once the dynamic routing is created, you can modify the associated decision matrix by clicking **Edit Dynamic Rules**. This launches the Rule Designer and enables modification of the associated decision table of the Business Rules component. After you create dynamic routing for the Mediator component, you cannot go back to static routing without deleting the dynamic routing. Currently, there is no option for mixing these two types of routing.

The Mediator `mp1` file looks like the following after dynamic routing option is chosen:

Figure 19–35 Mediator `mp1` File for a Mediator with Dynamic Routing Rule



You see the following changes in the source view:

```
<Mediator name="Shipment" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://xmlns.oracle.com/sca/1.0/mediator">
  <operation name="execute" deliveryPolicy="AllOrNothing" priority="0">
    <switch decisionServiceRef="Phase1DecisionService"
      decisionServiceOperation="executeFunction"></switch>
  </operation>
</Mediator>
```

The `switch` element contains the decision service reference and operation details to enable the Mediator component to invoke the decision service in runtime for getting the dynamic routing decisions. Dynamic decisions are returned by rule engine user configuration in Runtime.

External service invocation contains an extra attribute called **bindingInfo**, which contains binding information to make the invocation dynamic.

Limitations on Mediators Using Dynamic Routing Rules

Following are some limitations on Mediators using dynamic routing rules:

- As of now, only SOAP bindings are supported. There will be a dummy SOAP binding in the `composite.xml` file. This endpoint will be overridden by Mediator in runtime through NM property. So, outbound services can be called only over SOAP.
- Payload manipulation is limited for dynamic routing rules. No assignment, transformation, or validation can be performed.
- The reference WSDL file (Layer 2 or Called References) should have the same abstract WSDL file as the Phase Reference that gets auto created.
- Dynamic Routing is not possible for Mediators with synchronous or one-way interface.

19.3 Creating a Mediator for Routing Messages

The CustomerRouter use case provides an overview of how to use a Mediator in a SOA composite sample application to route messages the payload. For downloading the sample files mentioned in this section, visit the following URL:

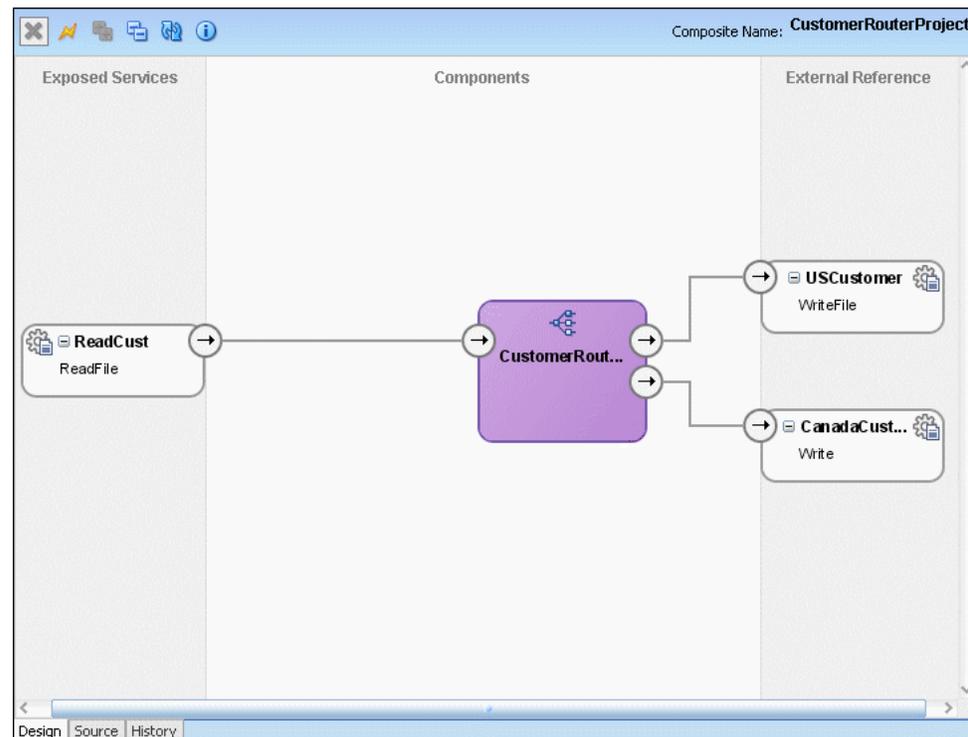
http://www.oracle.com/technology/sample_code/products/mediator

The CustomerRouter use case consists of the following steps:

1. Legacy customer files are picked up from a directory by an adapter service named ReadCust.
2. The ReadCust adapter service sends the file data to the CustomerRouter Mediator.
3. The CustomerRouter Mediator applies a filter to the XML message payload to determine whether the message should be routed to the USCustomer reference or CanadaCustomer reference.
4. The CustomerRouter Mediator then transforms the message to the structure required by the adapter reference.
5. The external reference delivers the message to its associated external application.

Figure 19–36 provides an overview of the CustomerRouter use case.

Figure 19–36 Overview of CustomerRouter Use Case



19.3.1 Step-By-Step Instructions for Creating the CustomerRouter Use Case

This section provides the design-time tasks for creating, building, and deploying the use case. These tasks should be performed in the order in which they are presented.

- Section 19.3.1.1, "Task 1: Creating an Oracle JDeveloper Application and Project"

- [Section 19.3.1.2, "Creating CustomerRouter Mediator Component"](#)
- [Section 19.3.1.3, "Creating a File Adapter Service"](#)
- [Section 19.3.1.4, "Creating a File adapter reference"](#)
- [Section 19.3.1.5, "Specifying Routing Rules"](#)
- [Section 19.3.1.6, "Creating Oracle Application Server Connection"](#)
- [Section 19.3.1.7, "Deploying CustomerRouterProject"](#)

19.3.1.1 Task 1: Creating an Oracle JDeveloper Application and Project

To create an application and a project for the use case:

1. In Oracle JDeveloper, click **File** and select **New**.
The New Gallery dialog appears.
2. In the New Gallery, expand the **General** node, and select the **Applications** category.
3. In the **Items** list, select **SOA Application** and click **OK**.
The Create SOA Application Wizard appears.
4. In the **Application Name** field, enter `CustomerRouter` and then click **Next**.
The Name your project screen appears.
5. In the **Project Name** field, enter `CustomerRouterProject` and click **Next**.
The Configure SOA settings screen appears.
6. In the Composite Template list, select **Empty Composite** and then click **Finish**.
The Applications Navigator of Oracle JDeveloper is populated with the new application and the project, and the Design tab contains a blank palette.
7. From the **File** menu, click **Save All**.

19.3.1.2 Creating CustomerRouter Mediator Component

To create a Mediator named CustomerRouter:

1. From the Component Palette, select **SOA**.
2. Drag and drop a **Mediator** to the Components design area.
The Create Mediator dialog is displayed.
3. Enter `CustomerRouter` in the **Name** field.
4. Select **Define Interface Later** from Templates.
5. Click **OK**.
A Mediator with name `CustomerRouter` is created.

19.3.1.3 Creating a File Adapter Service

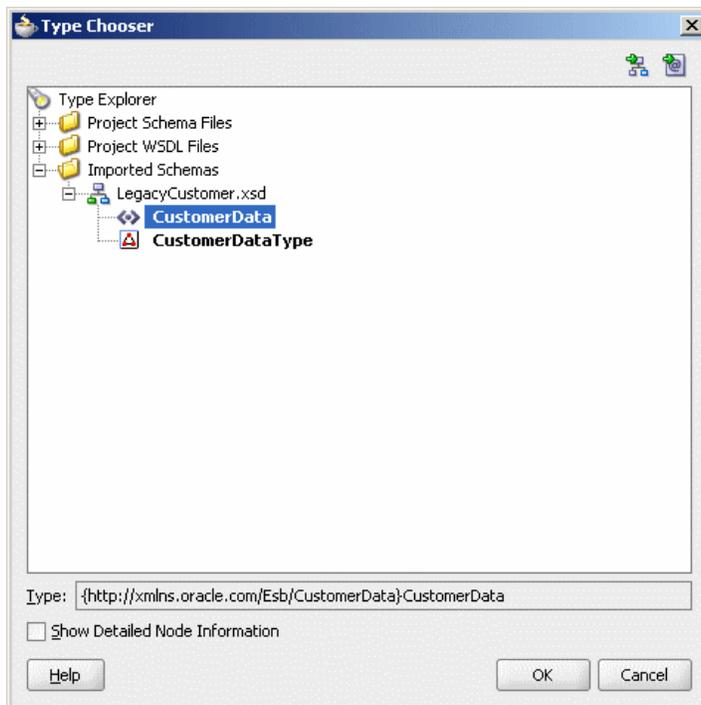
You must create a File adapter service named `ReadCust` to read the XML files from a directory.

Note: Mediator may process the same file twice when run against Oracle RAC planned outages. This is because a File adapter is a non-XA compliant adapter. So, when it participates in a global transaction, it may not follow the XA interface specification of processing each file once and only once.

To create a File adapter service:

1. From the Components Palette, select **SOA**.
2. Select **File Adapter** and drag it to the Exposed Services design area.
The Adapter Configuration Wizard Welcome page is displayed.
3. Click **Next**.
The Service Name page is displayed.
4. In the **Service Name** field, enter `ReadCust`.
5. Click **Next**.
The Adapter Interface page is displayed.
6. Select **Define from operation and schema (specified later)** and click **Next**.
The Operation page is displayed.
7. In the **Operation Type** field, select **Read File**.
8. In the **Operation Name** field, replace **Read** with `ReadFile`.
9. Click **Next**.
The File Directories page is displayed.
10. In the **Directory for Incoming Files (physical path)** field, enter the directory from which you want to read the files. For example, `C:\Customer\In`.
11. Click **Next**.
The File Filtering page is displayed.
12. In the **Include Files with Name Pattern** field, enter `*.xml`, and then click **Next**.
The File Polling page is displayed.
13. Change the **Polling Frequency** field value to **10 seconds**, and then click **Next**.
The Messages page is displayed.
14. Click **Search** to the right of the **URL** field.
The Type Chooser dialog is displayed.
15. Click **Import Schema File**.
The Import Schema File dialog is displayed.
16. Click **Search** to the right of the **URL** field and select the `LegacyCustomer.xsd` file present in the `Samples` folder.
17. Click **OK**.
18. Expand the navigation tree to **Type Explorer\Imported Schemas\LegacyCustomer.xsd** and select **CustomerData**, as shown in [Figure 19-37](#).

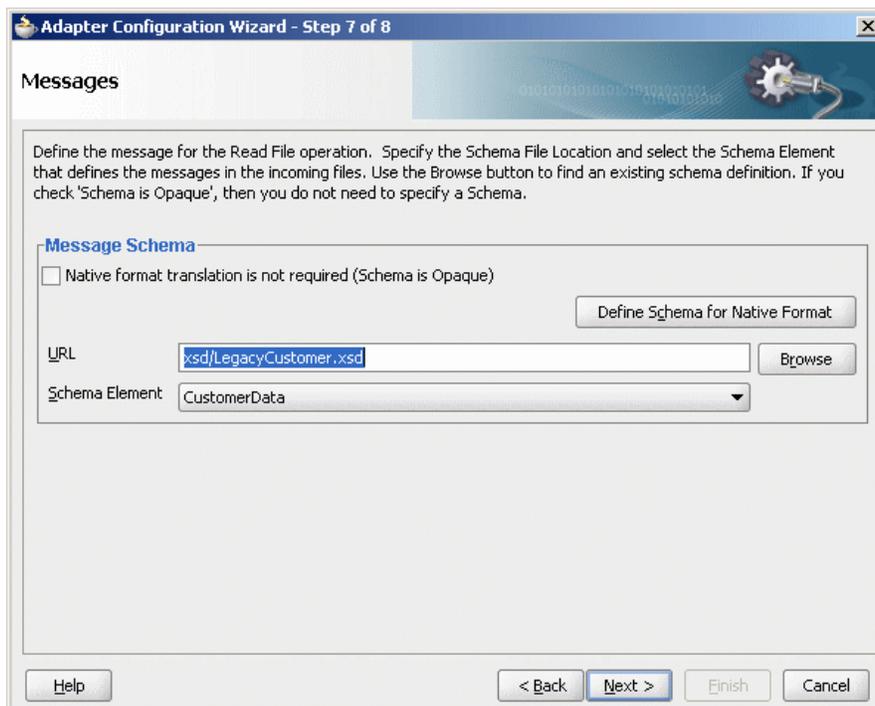
Figure 19–37 Type Chooser - CustomerData



19. Click **OK**.

The Adapter Configuration wizard appears as shown in [Figure 19–38](#).

Figure 19–38 Adapter Configuration Wizard – Messages page



20. Click **Next**.

The Finish page is displayed.

21. Click **Finish**.
22. From the **File** menu, click **Save All**.

19.3.1.4 Creating a File adapter reference

You must create a File adapter reference `USCustomer`.

To create a File adapter reference:

1. From the Component Palette, select **SOA**.
2. Select **File Adapter** and drag it to the External References design area.
The Adapter Configuration Wizard Welcome page is displayed.
3. Click **Next**.
The Service Name page is displayed.
4. In the **Service Name** field, enter `USCustomer`.
5. Click **Next**.
The Adapter Interface page is displayed.
6. Select **Define from operation and schema (specified later)** and click **Next**.
The Operation page is displayed.
7. Click **Next**.
The Operation page is displayed.
8. In the **Operation Type** field, select **Write File**.
9. In the **Operation Name** field, enter `WriteFile`.
10. Click **Next**.
The File Configuration page is displayed.
11. In the **Directory for Outgoing Files (physical path)** field, enter the name of the directory where you want to write the files.
For example, `C:\Customer\out`.
12. In the **File Naming Convention** field, enter `customer_%SEQ%.xml` and click **Next**.
The Messages page is displayed.
13. Click **Search** to the right of the **URL** field.
The Type Chooser dialog is displayed.
14. Click **Import Schema File**.
The Import Schema File dialog is displayed.
15. Click **Search** to the right of the **URL** field and select the `USCustomer.xsd` file present in the `Samples` folder.
16. Click **OK**.
17. Expand the navigation tree to **Type Explorer\Imported Schemas\USCustomer.xsd** and then select **Customer**.
18. Click **OK**.

19. Click Next.

The Finish page is displayed.

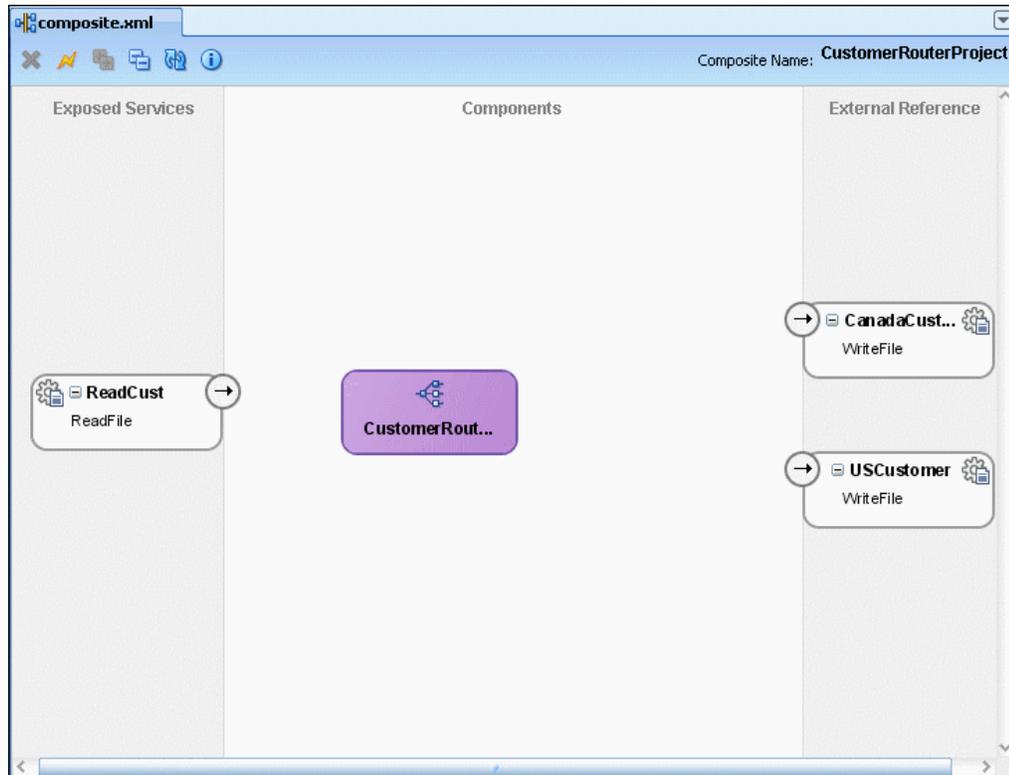
20. Click Finish.

21. From the File menu, click Save All.

Create another File adapter reference CanadaCustomer in similar way by using the CanCustomer.xsd file.

Figure 19–39 shows how the SOA composite editor appears after performing this task.

Figure 19–39 Mediator Component with Adapter Services and References



19.3.1.5 Specifying Routing Rules

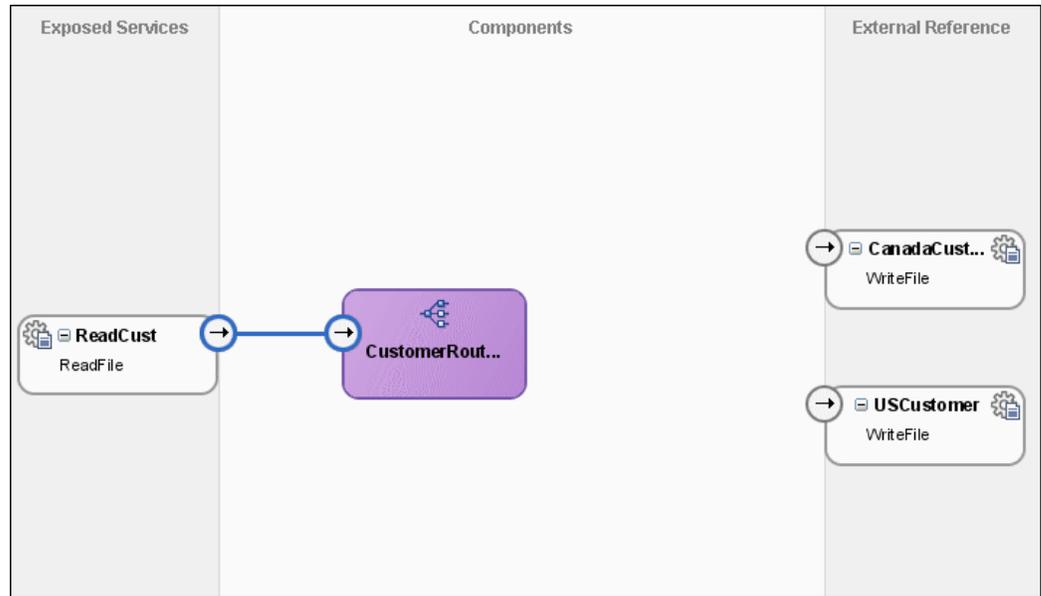
You must specify the path that messages take from the ReadCust adapter service to external references.

To specify routing rules:

1. Connect the ReadCust service to the CustomerRouter Mediator as shown in Figure 19–40.

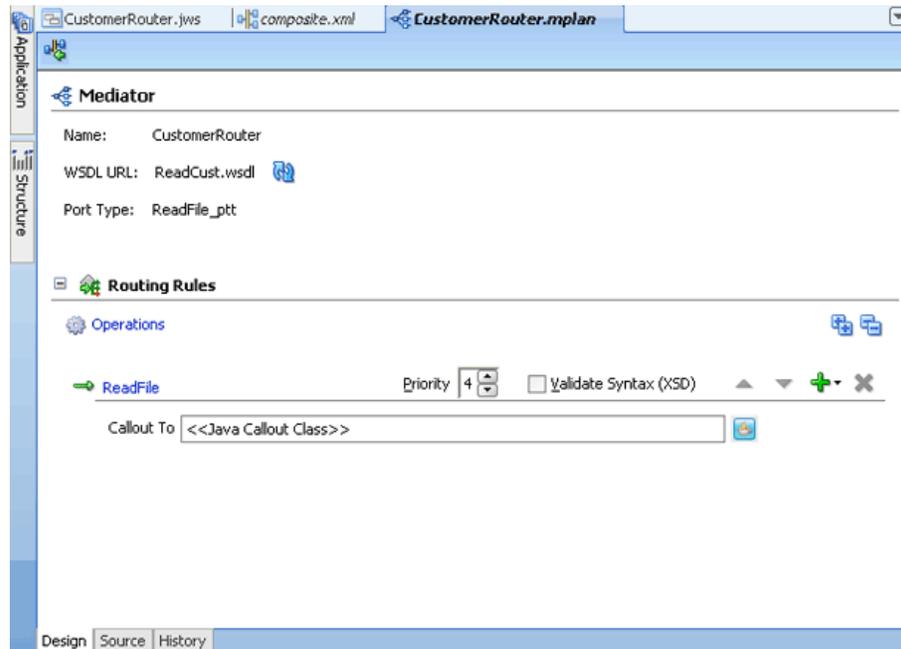
This specifies the File adapter service to invoke the CustomerRouter Mediator while reading a file from the input directory.

Figure 19–40 Connecting ReadCust Service to the CustomerRouter Mediator



2. Double-click **CustomerRouter** Mediator to open the `CustomerRouter.mplan` editor shown in [Figure 19–41](#).

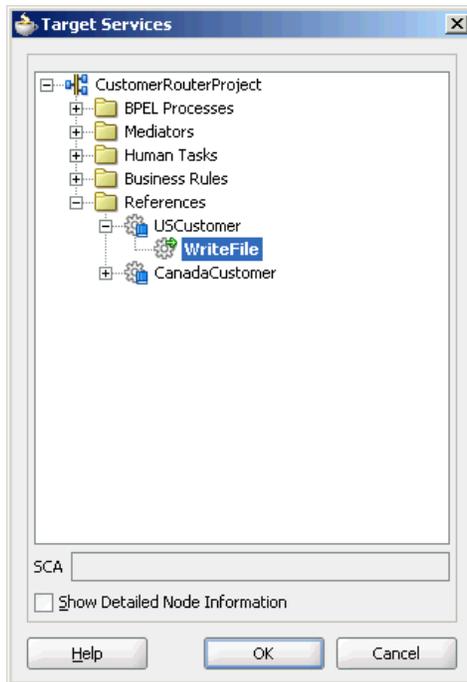
Figure 19–41 CustomerRouter Mediator in Mediator Editor



3. In the Routing Rules section, click **Add** to the extreme right side of `ReadFile` and then click **static routing rule**.
The Target Type dialog is displayed.
4. Click **Service**.
The Target Services dialog is displayed.

5. Navigate to **CustomerRouterProject**, **References**, **USCustomer** and select **WriteFile** as shown in [Figure 19–42](#).

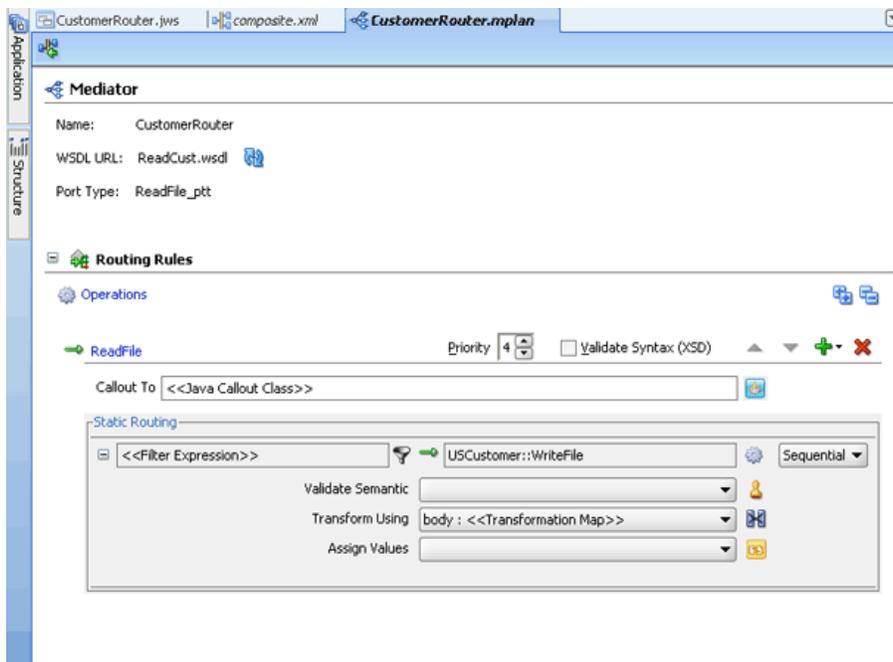
Figure 19–42 Target Services Dialog



6. Click **OK**.

The Routing Rules panel is displayed, as shown in [Figure 19–43](#).

Figure 19–43 The Routing Rules Panel - MapCustomerData Added



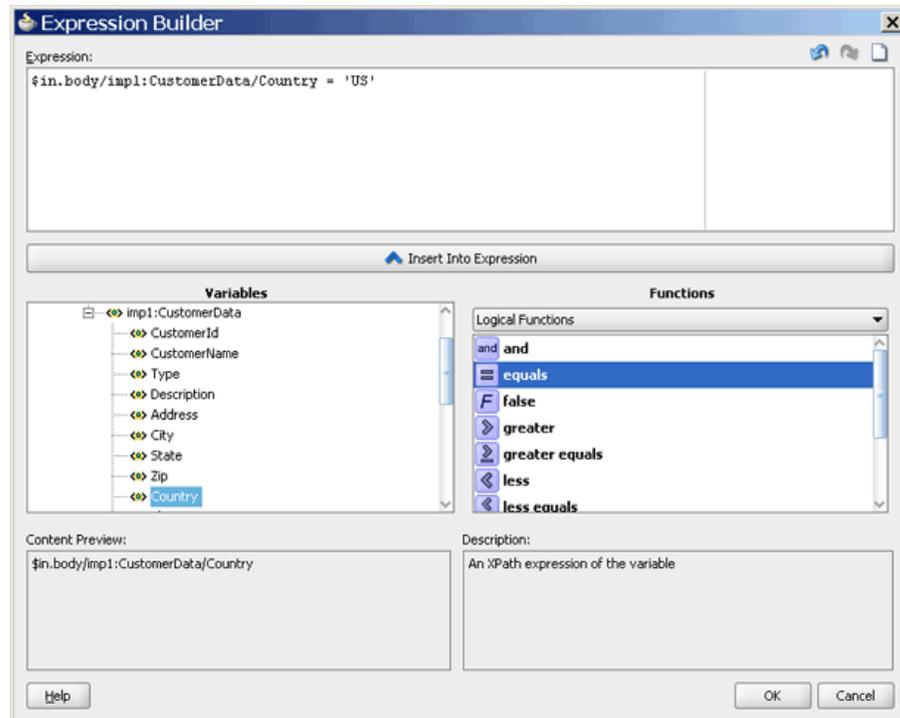
- Click the **filter** icon next to the <<Filter Expression>> field to create a filter expression for this routing rule.

The Expression Builder dialog is displayed.

- In the **Variables** field, navigate to **Variables, in, body, imp1:CustomerData**, and then select **Country**.
- Double-click **Country**.

The Country node is added in the **Expression** field as shown in [Figure 19-44](#).

Figure 19-44 Expression Builder Dialog



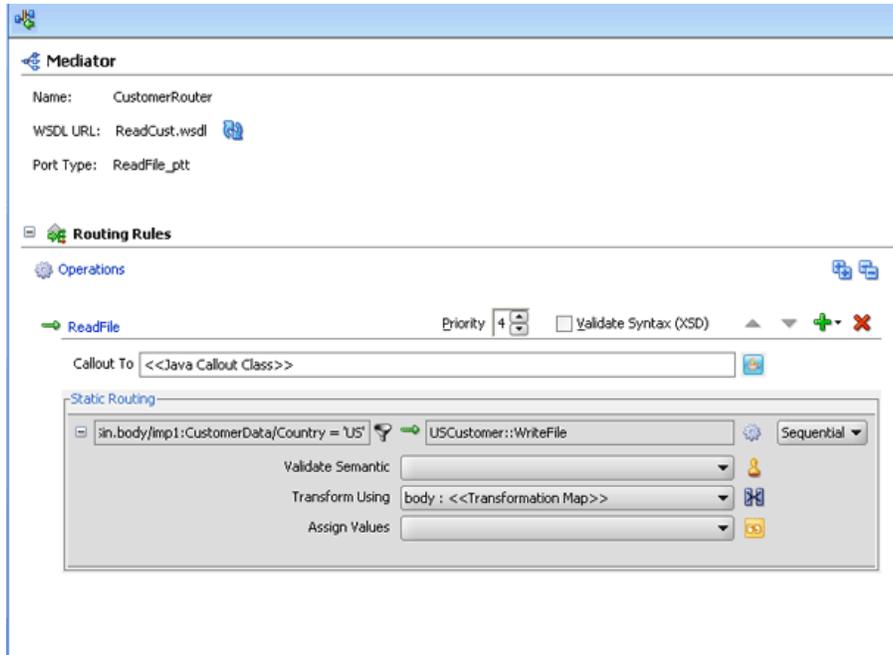
- Modify the expression to the following:

```
$in.CustomerData/imp1:CustomerData/Country='US'
```

- Click **OK**.

The <<Filter Expression>> field of the Routing Rules panel is populated with the expression as shown in [Figure 19-45](#).

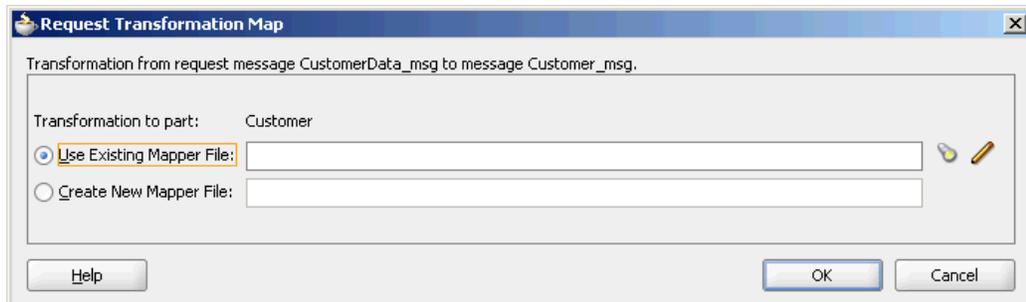
Figure 19–45 Populated Filter Field of Routing Rules Panel



12. Click the icon to the right of the **Transform Using** field.

The Request Transformation Map dialog is displayed, as shown in [Figure 19–46](#).

Figure 19–46 Request Transformation Map



13. Select **Create New Mapper File** and click **OK**.

A CustomerData_To_Customer.xsl tab is added, as shown in [Figure 19–47](#).

Figure 19–47 CustomerData_To_Customer.xsl Tab – Initially



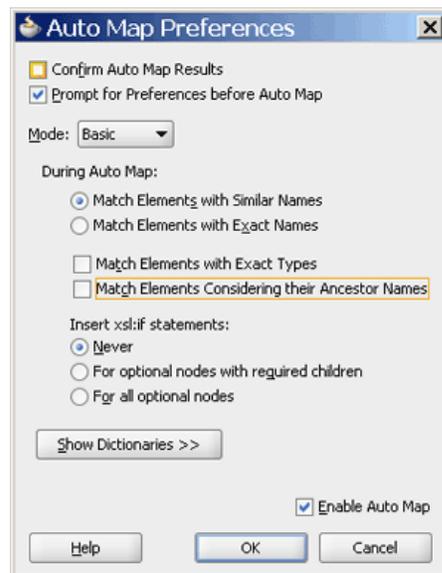
14. Drag and drop the **imp1:CustomerData** source element to **imp1:Customer** target element.

The Auto Map Preferences dialog is displayed.

15. From the **During Auto Map** options, deselect **Match Elements Considering their Ancestor Names**.

The Auto Map Preferences dialog is shown in [Figure 19–48](#).

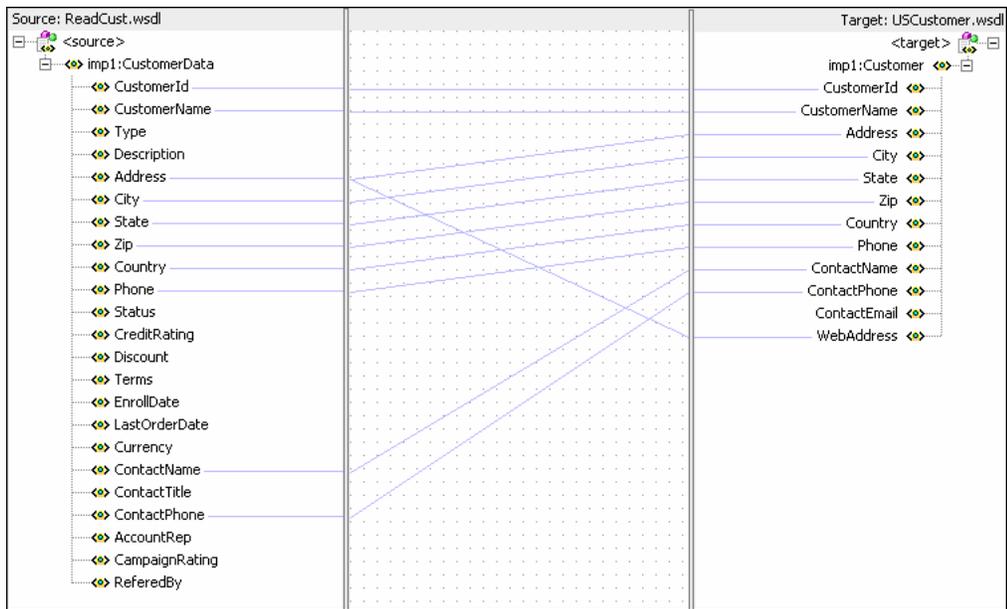
Figure 19–48 Auto Map Preferences Dialog



16. Click **OK**.

The CustomerData_To_Customer.xsl tab appears as shown in [Figure 19–49](#).

Figure 19–49 CustomerData_To_Customer.xsl Tab – Auto Mapped Connections

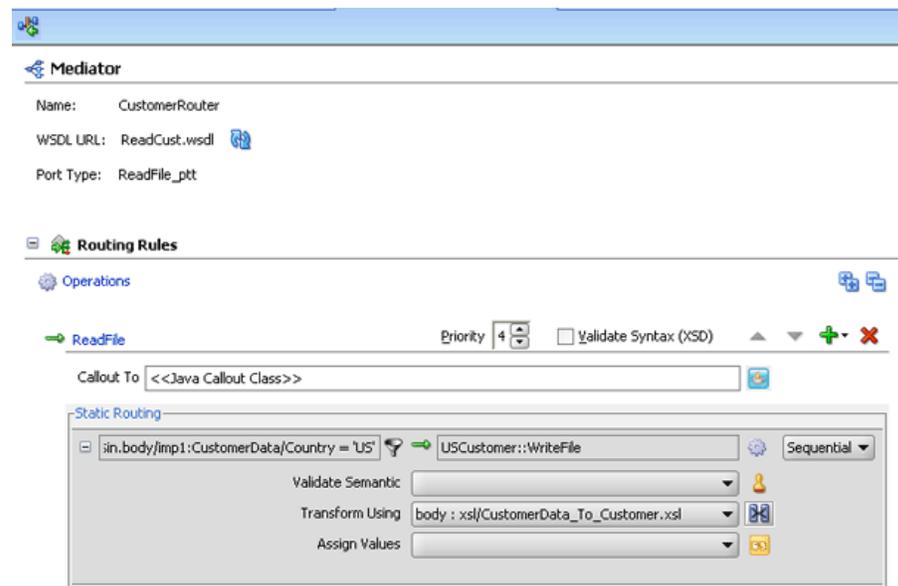


17. From the **File** menu, click **Save All**.
18. Repeat the steps mentioned in Step 3 through 17 to specify `CanadaCustomer` reference as the target service. In the Expression builder dialog, specify the following expression:

Note: For repeating the steps, you must reenter the Mediator Editor by closing the Mapper Editor or by clicking the `CustomerRouter.mplan` tab.

```
$in.CustomerData/imp1:CustomerData/Country='CA'
```

Figure 19–50 shows how the Mediator editor would appear after you have specified `CanadaCustomer` reference as target service.

Figure 19–50 Routing Rules Panel with Target Services Defined

After performing all the steps mentioned in this section, the SOA composite editor would appear as shown in [Figure 19–36](#).

19.3.1.6 Creating Oracle Application Server Connection

An Oracle Application Server connection is required for deploying your SOA composite application. For information on creating Oracle Application Server connection, refer to *Oracle Fusion Middleware User's Guide for Technology Adapters*.

19.3.1.7 Deploying CustomerRouterProject

Deploying the `CustomerRouterProject` composite application to Oracle Application Server consists of following steps:

- Creating an Application Deployment Profile
- Deploying the Application Deployment Profile to Oracle Application Server

For detailed information about these steps, see [Section 43.2, "Deploying a Single SOA Composite in Oracle JDeveloper"](#).

19.3.2 Running and Monitoring the CustomerRouterProject Application

After deploying the `CustomerRouterProject` application, you can run it by copying the input xml files to the input folder. The payload, the files will be written to the specified output directories.

For monitoring the running instance, you can use the Oracle Enterprise Manager Console at the following URL:

`http://hostname:portnumber/em`

where *hostname* is the host on which you installed the Oracle SOA Suite infrastructure and *portnumber* is the port of the server, where Enterprise Manager is installed.

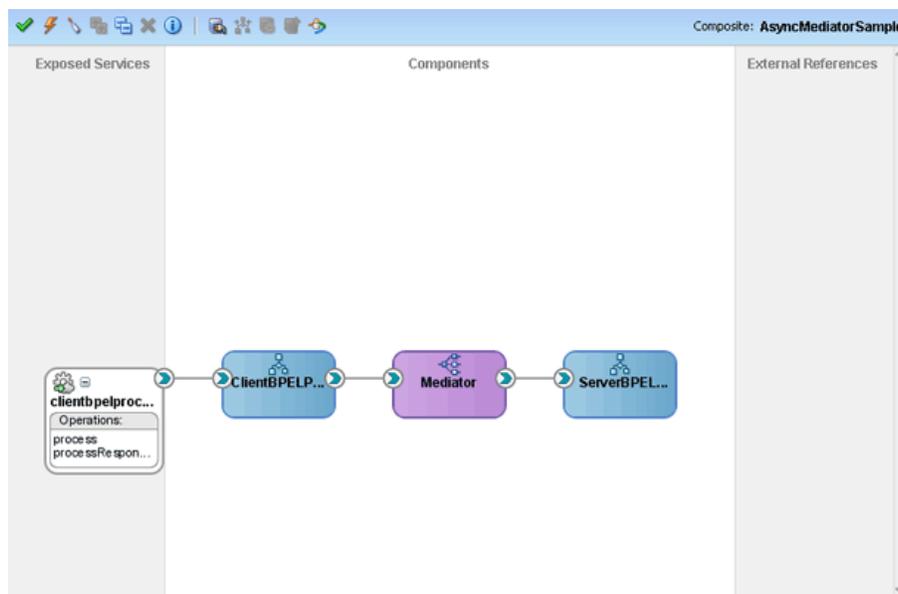
For detailed information about these steps, see [Section 43.2, "Deploying a Single SOA Composite in Oracle JDeveloper"](#).

19.4 Creating Asynchronous Request Response Using Mediator

This sample demonstrates asynchronous request response scenario using Mediator. This composite has a client BPEL process invoking a Mediator, which invokes a server BPEL process. All the invocations are done as asynchronous request response.

[Figure 19–51](#) provides an overview of the AsyncMediator use case.

Figure 19–51 Overview of AsyncMediator Use Case



For downloading the sample files mentioned in this section, visit the following URL:

http://www.oracle.com/technology/sample_code/products/mediator

19.4.1 Step-By-Step Instructions for Creating the AsyncMediator Use Case

This section provides the design-time tasks for creating, building, and deploying the use case. These tasks should be performed in the order in which they are presented.

- [Section 19.4.1.1, "Task 1: Creating an Oracle JDeveloper Application and Project"](#)
- [Section 19.4.1.2, "Task 2: Creating a Server BPEL Process"](#)
- [Section 19.4.1.3, "Task 3: Create a Mediator Component"](#)
- [Section 19.4.1.4, "Task 4: Creating a Client BPEL Process"](#)
- [Section 19.4.1.5, "Task 5: Creating the Invoke, Receive, and Assignment Activities"](#)
- [Section 19.4.1.6, "Task 6: Configuring Oracle Application Server Connection"](#)
- [Section 19.4.1.7, "Task 7: Deploying the Composite Application"](#)

19.4.1.1 Task 1: Creating an Oracle JDeveloper Application and Project

To create an application and a project for the use case:

1. In Oracle JDeveloper, click **File** and select **New**.
The New Gallery dialog appears.
2. In the New Gallery, expand the **General** node, and select the **Applications** category.
3. In the **Items** list, select **SOA Application** and click **OK**.
The Create SOA Application Wizard appears.
4. In the **Application Name** field, enter `AsyncMediator` and then click **Next**.
The Name your project screen appears.
5. In the **Project Name** field, enter `AsyncMediatorSample` and click **Next**.
The Configure SOA settings screen appears.
6. In the Composite Template list, select **Empty Composite** and then click **Finish**.
The Applications Navigator of Oracle JDeveloper is populated with the new application and the project, and the Design tab contains a blank palette.
7. From the **File** menu, click **Save All**.

19.4.1.2 Task 2: Creating a Server BPEL Process

To create a server BPEL process:

1. In the Application Navigator, double-click **composite.xml**. The `composite.xml` window is displayed.
2. From the Component Palette, select **SOA**.
3. Drag and drop a BPEL process to the Components design area.
The Create BPEL Process dialog is displayed.
4. In the **Name** field, enter `ServerBPELProcess`.
5. In the **Template** field, select **Asynchronous BPEL Process**.
6. Uncheck **Expose as a SOAP service** and click **OK**. The `ServerBPELProcess` is created in the `composite.xml` window.

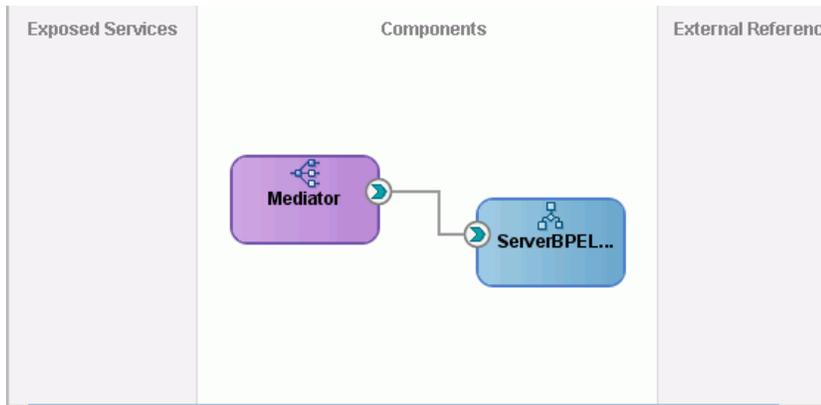
19.4.1.3 Task 3: Create a Mediator Component

To create a Mediator named Mediator:

1. From the Component Palette, select **SOA**.
2. Drag and drop a **Mediator** to the Components design area.
The Create Mediator dialog is displayed.
3. Enter `Mediator` in the **Name** field and select **Asynchronous Interface** from Template.
4. Uncheck **Create Composite Service with SOAP Bindings**.
5. Click **OK**.

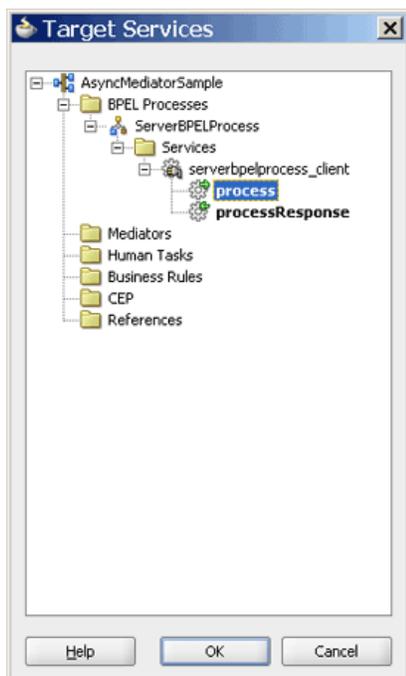
A Mediator with name `Mediator` is created, as shown in [Figure 19-52](#).

Figure 19–52 Mediator and ServerBPELProcess in the Composite Window



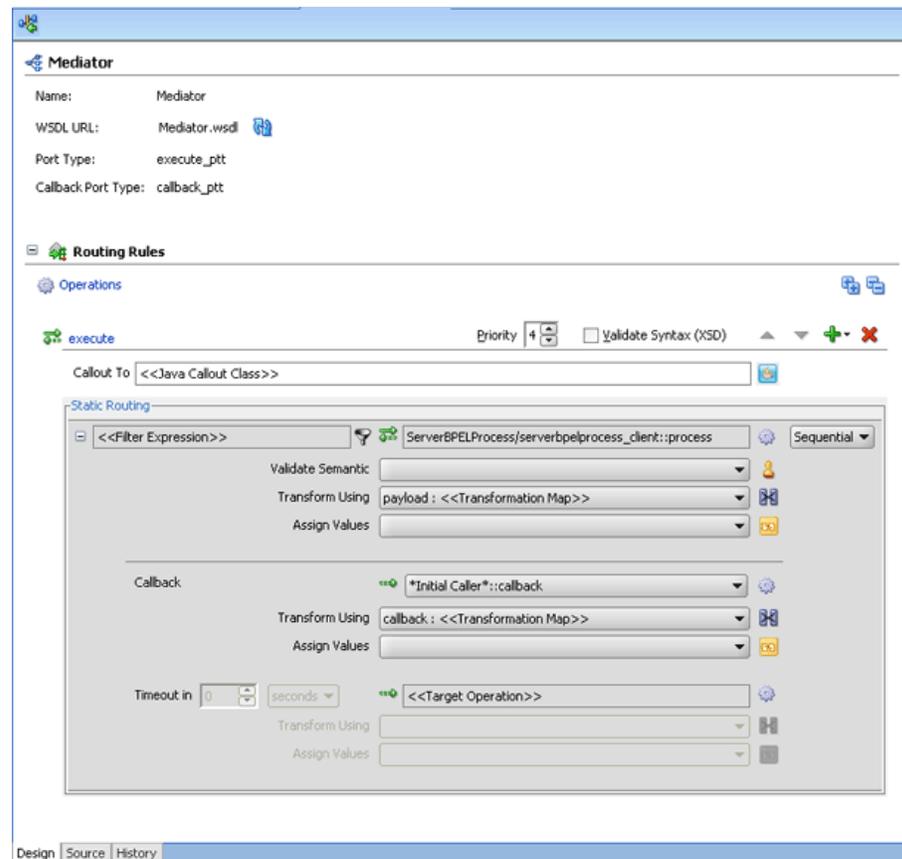
6. Double-click the **Mediator** Mediator.
The Mediator.mplan window is displayed.
7. In the Routing Rules section, click **Add** to the extreme right side of `execute` and then static routing rules.
The Target Type dialog is displayed.
8. Select **Service**.
The Target Services dialog is displayed.
9. Navigate to **AsyncMediatorSample**, **BPEL Processes**, **ServerBPELProcess**, **Services**, **serverbpelprocess_client**, and **process**, as shown in [Figure 19–53](#).

Figure 19–53 Target Services Dialog



10. Click **OK**.
The Routing Rules panel is displayed, as shown in [Figure 19–54](#).

Figure 19–54 The Routing Rules Panel - initiate Added



11. Click the icon to the right of the **Transform Using** field, below <<Filter Expression field>>.

The Request Transformation Map dialog is displayed.

12. Select **Create New Mapper File** and click **OK**.

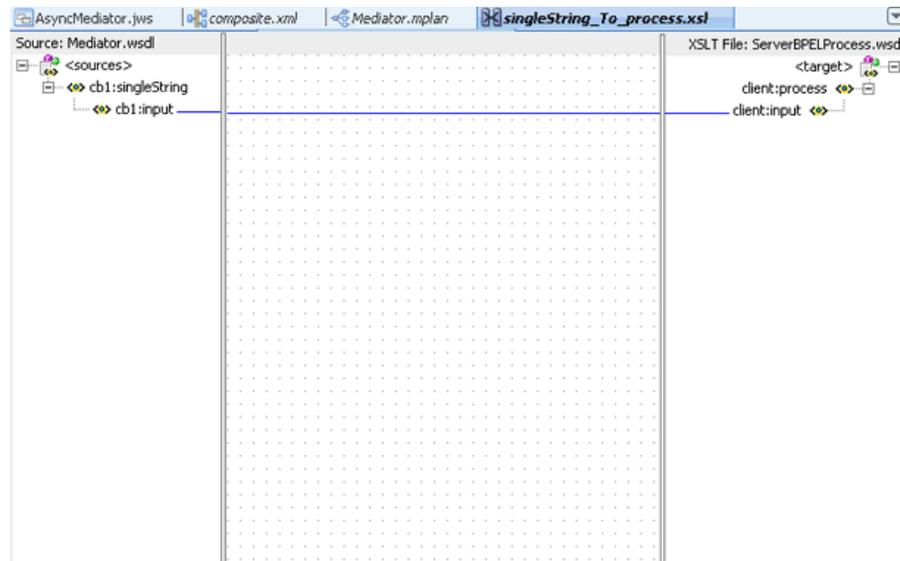
A singleString_To_process.xsl tab is added.

13. Drag and drop the **cb1:input** source element to **client:input** target element.

The Auto Map Preferences dialog is displayed.

14. From the **During Auto Map** options, deselect **Match Elements Considering their Ancestor Names** and click **OK**. The singleString_To_process.xsl window is displayed, as shown in Figure 19–55.

Figure 19–55 *singleString_To_process.xsl* Window



15. In the Routing Rules panel, under Callback, click the icon to the right of the **Transform Using** field.
The Request Transformation Map dialog is displayed.
16. Select **Create New Mapper File** and click **OK**.
A processResponse_To_singleString.xsl tab is added.
17. Drag and drop the **client:processResponse** source element to **cb1:singleString** target element.
The Auto Map Preferences dialog is displayed.
18. From the **During Auto Map** options, deselect **Match Elements Considering their Ancestor Names** and click **OK**.

19.4.1.4 Task 4: Creating a Client BPEL Process

To create a client BPEL Process:

1. In the Application Navigator, double-click **composite.xml**. The composite.xml window is displayed.
2. From the Component Palette, select **SOA**.
3. Drag and drop a BPEL process to the Components design area.
The Create BPEL Process dialog is displayed.
4. In the **Name** field, enter `ClientBPELProcess`.
5. In the **Template** field, select **Asynchronous BPEL Process**.
6. Click **OK**. The ClientBPELProcess is created in the composite.xml window.
7. Drag and drop the ClientBPELProcess BPEL process to the Mediator Mediator. The composite.xml appears as shown in [Figure 19–51](#).

19.4.1.5 Task 5: Creating the Invoke, Receive, and Assignment Activities

To create the invoke activity:

1. Double-click **ClientBPELProcess**. The ClientBPELProcess.bpel page is displayed.
2. Drag and drop an **Invoke** activity from the Component Palette to the design area.
3. Double-click the **Invoke** activity. The Invoke dialog is displayed.
4. Enter `InvokeMediator` in the **Name** field.
5. Click **Browse Partner Links** next to the **Partner Link** field. The Partner Link Chooser dialog is displayed.
6. Select **Operation - execute**, and click **OK**.
7. Click the **Auto-Create Variable** icon to the right of the **Variable** field in the Receive dialog. The Create Variable dialog is displayed.
8. Enter `InvokeMediator_execute_InputVariable_1` in the **Variable** field and click **OK**. The Invoke dialog is displayed.
9. Click **OK**. The Oracle JDeveloper ClientBPELProcess.bpel page appears.

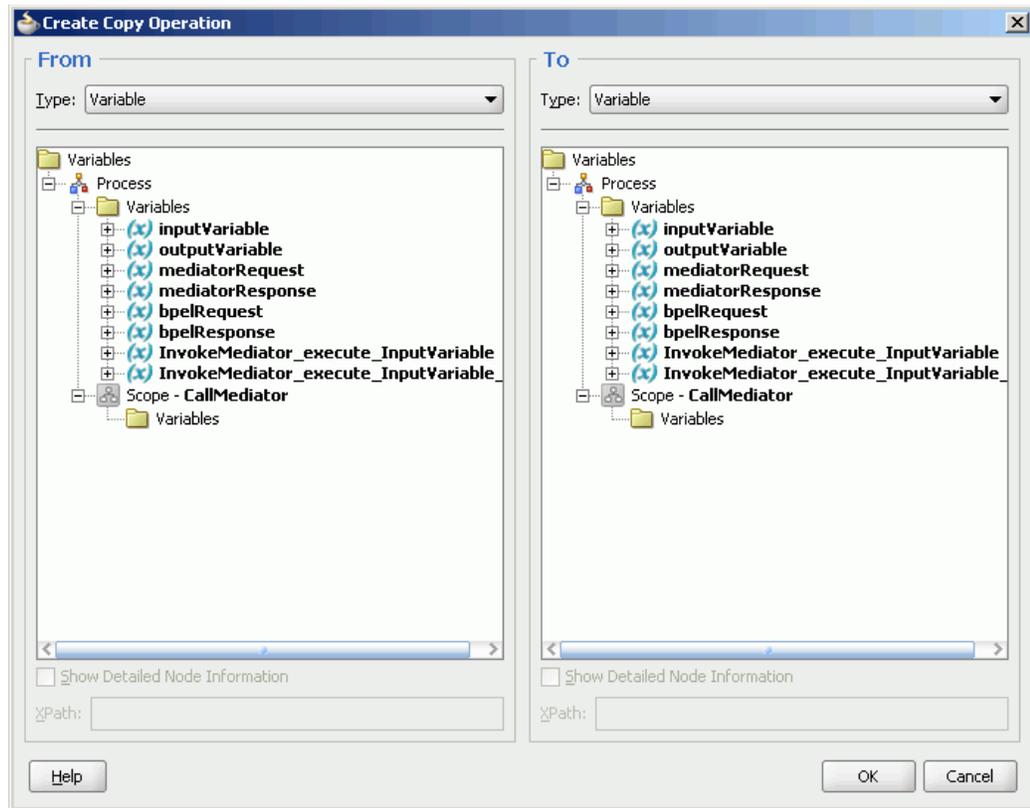
To create the receive activity:

1. Drag and drop a **Receive** activity from the Component Palette to the design area.
2. Double-click the **Receive** activity. The Receive dialog is displayed.
3. Enter `ReceiveFromMediator` in the **Name** field.
4. Click **Browse Partner Links** next to the **Partner Link** field. The Partner Link Chooser dialog is displayed.
5. Select **Operation - callback**, and click **OK**.
6. Click the **Auto-Create Variable** icon to the right of the **Variable** field in the Receive dialog. The Create Variable dialog is displayed.
7. Select the default variable name and click **OK**. The **Variable** field is populated with the default variable name.
8. Check **Create Instance**, and click **OK**. The Oracle JDeveloper ClientBPELProcess.bpel page appears.

To create the assignment activity:

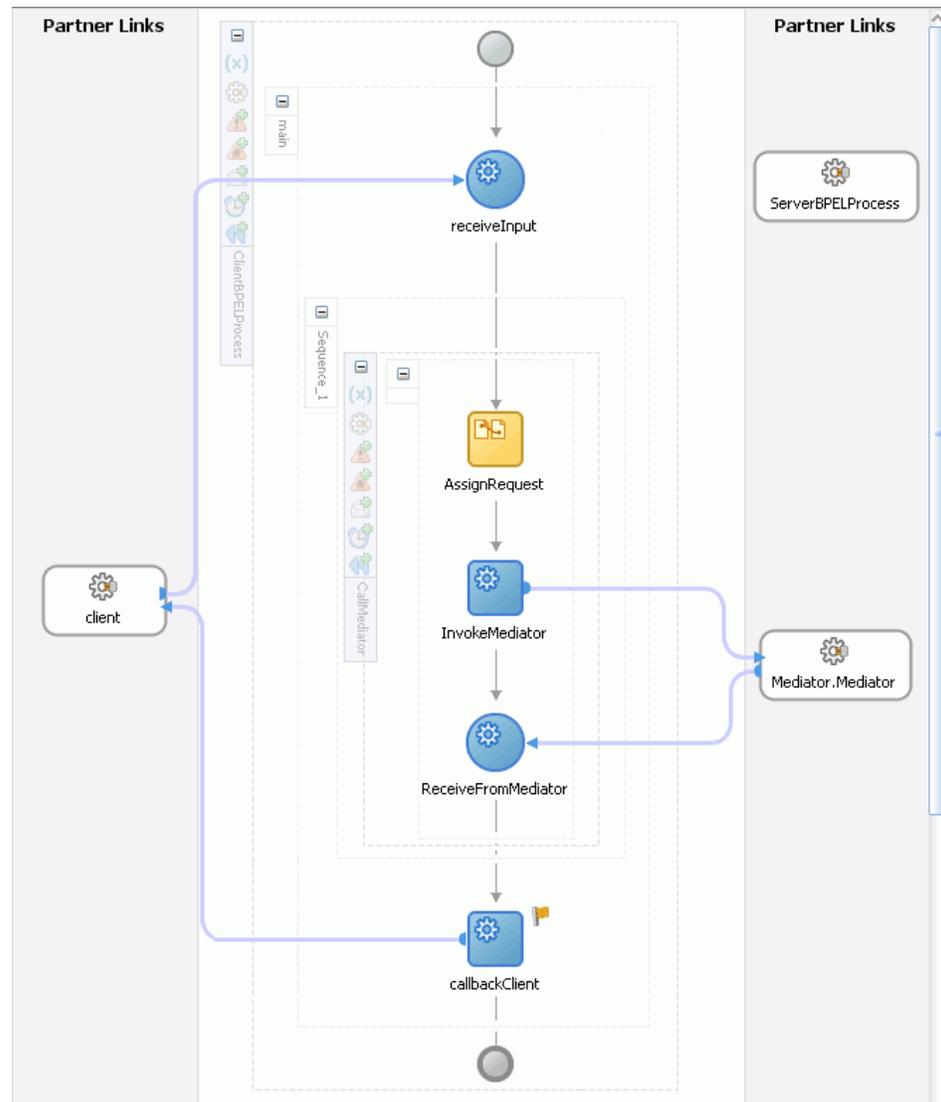
1. Drag and drop an **Assign** activity from the Component Palette between the `ReceiveFromMediator` and `InvokeMediator` activities in the design area.
2. Double-click the **Assign** activity. The Assign dialog is displayed.
3. Enter `AssignRequest` in the **Name** field.
4. Click the **Copy Operation** tab. The Assign dialog is displayed.
5. Select **Copy Operation**. The Create Copy Operation dialog is displayed.
6. Create the copy operation between the triggers file name and the file variable, as shown in [Figure 19-56](#).

Figure 19–56 The Create Copy Operation Dialog



7. Click **OK** in the Create Copy Operation dialog.
8. Click **OK** to return to the Oracle JDeveloper ClientBPELProcess.bpel page, as shown in [Figure 19–57](#).

Figure 19–57 The Oracle JDeveloper - ClientBPELProcess.bpel

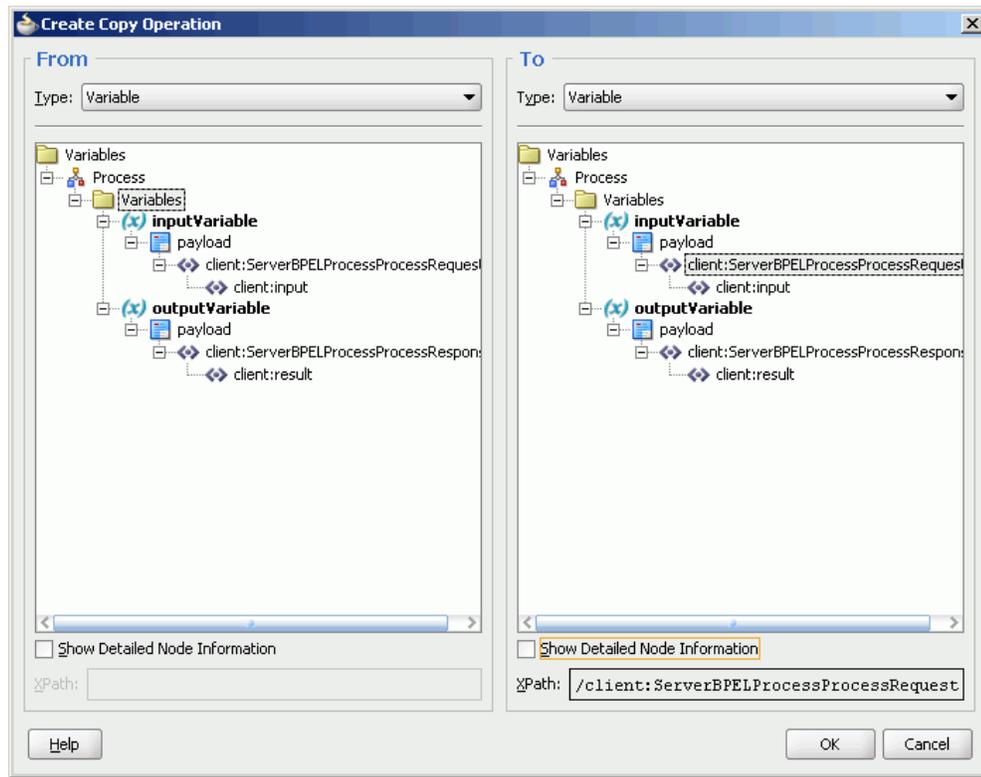


9. Click **File, Save All**.

To create an assign activity in the ServerBPELProcess.bpel Window

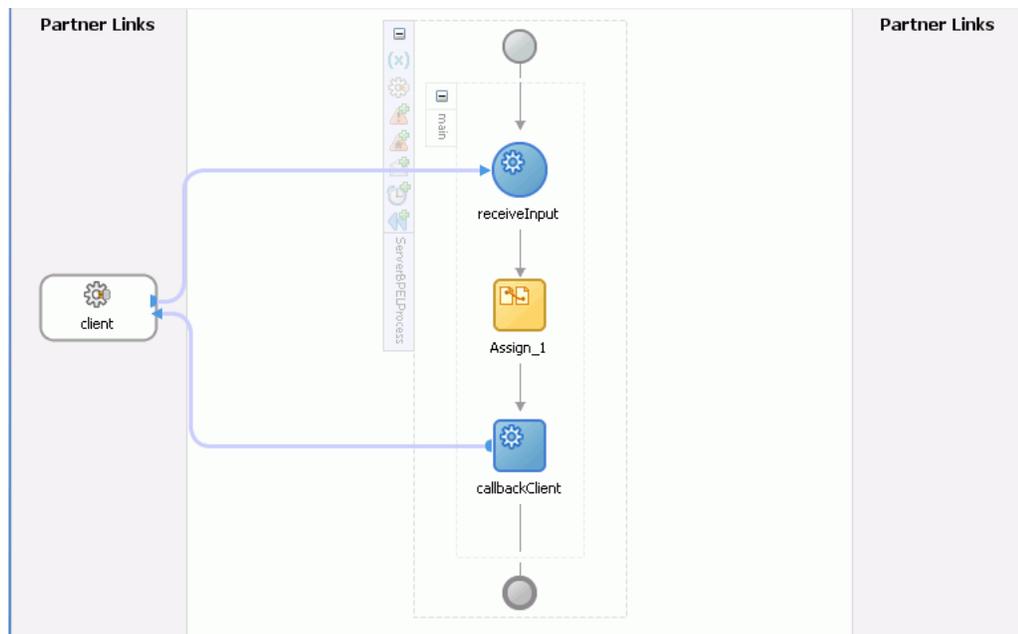
1. Double-click the **ServerBPELProcess.bpel** BPEL process. The ServerBPELProcess.bpel window is displayed.
2. Drag and drop an **Assign** activity from the Component Palette between the **receiveInput** and **callbackClient** activities in the design area.
3. Double-click the **Assign** activity. The Assign dialog is displayed.
4. Click the **Copy Operation** tab. The Assign dialog is displayed.
5. Select **Copy Operation**. The Create Copy Operation dialog is displayed.
6. Create the copy operation between the triggers file name and the file variable, as shown in [Figure 19–58](#).

Figure 19–58 The Create Copy Operation Dialog



7. Click **OK** in the Create Copy Operation dialog.
8. Click **OK** to return to the Oracle JDeveloper ServerBPELProcess.bpel page, as shown in [Figure 19–59](#).

Figure 19–59 The Oracle JDeveloper - ServerBPELProcess.bpel



9. Click **File, Save All**.

19.4.1.6 Task 6: Configuring Oracle Application Server Connection

An Oracle Application Server connection is required for deploying your SOA composite application. For information on creating Oracle Application Server connection, refer to [Section 19.3.1.6, "Creating Oracle Application Server Connection"](#).

19.4.1.7 Task 7: Deploying the Composite Application

Deploying the `EventMediatorApp` composite application to Oracle Application Server consists of following steps:

- Creating an Application Deployment Profile
- Deploying the Application to Oracle Application Server

For detailed information about these steps, see [Section 43.2, "Deploying a Single SOA Composite in Oracle JDeveloper"](#).

Using Mediator Error Handling

This chapter describes how to handle errors with Oracle Mediator (Mediator).

This chapter includes the following sections:

- [Section 20.1, "Introduction to Oracle Mediator Error Handling"](#)
- [Section 20.2, "Using Error Handling with Mediator"](#)
- [Section 20.3, "Fault Recovery Using Enterprise Manager"](#)
- [Section 20.4, "Error Handling XML Schema Definition Files"](#)

20.1 Introduction to Oracle Mediator Error Handling

Oracle Mediator provides sophisticated error handling capabilities that enable you to configure a Mediator service component for error occurrences and corresponding corrective actions. Error handling enables a Mediator to handle errors that occur during the processing of messages and also the exceptions returned by outside Web services. You can handle both business faults and system faults with Mediator.

Business faults are application-specific and are explicitly defined in the service WSDL file. You can handle business faults by defining the fault handlers in Oracle JDeveloper at design time. System faults occur because of some problem in the underlying system such as network not being available. Mediator provides fault policy-based error handling for system faults.

Fault policies enable you to handle errors automatically or through human intervention. Mediator fault policy-based error handling consists of the following three components:

- [Section 20.1.1, "Fault Policies"](#)
- [Section 20.1.2, "Fault Bindings"](#)
- [Section 20.1.3, "Error groups in Mediator"](#)

20.1.1 Fault Policies

A fault policy defines error conditions and corresponding actions. Fault policies are defined in the `fault-policies.xml` file. The `fault-policies.xml` file should be created based on the XML schema defined in [Section 20.4.1, "Schema Definition File for Fault-policies.xml"](#).

Note: Fault policies are applicable to parallel routing rules only. For sequential routing rules, the fault goes back to the caller and it is the responsibility of the caller to handle the fault. If the caller is an adapter, then you can define rejection handlers on the inbound adapter to take care of the errored out messages, that is, the rejected messages. For more information about Rejection Handlers, refer to *Oracle Fusion Middleware User's Guide for Technology Adapters*.

A sample fault policy file is shown in the following example:

```
<?xml version="1.0" encoding="UTF-8"?>
<faultPolicies>
  <faultPolicy version="2.0.1" id="CRM_ServiceFaults">
    <Conditions>
      <faultName xmlns:medns="http://schemas.oracle.com/mediator/faults"
name="medns:mediatorFault">
        <condition>
          <test>contains($fault.mediatorErrorCode, "TYPE_FATAL_MESH")</test>
          <action ref="ora-retry"/>
        </condition>
      </faultName>
    </Conditions>
    <Actions>
      <Action id="ora-retry">
        <retry>
          <retryCount>3</retryCount>
          <retryInterval>2</retryInterval>
          <exponentialBackoff/>
          <retryFailureAction ref="ora-java"/>
          <retrySuccessAction ref="ora-terminate"/>
        </retry>
      </Action>
    </Actions>
  </faultPolicy>
</faultPolicies>
```

The two components of the fault policy are described in the following sections:

- [Section 20.1.1.1, "Conditions"](#)
- [Section 20.1.1.2, "Actions"](#)

20.1.1.1 Conditions

Conditions identify error or fault conditions along with reference to the actions to be taken. You can use conditions to identify the action to be taken when a particular error or fault condition occurs. For example, for a particular error occurring because of a service not being available, you can perform an action such as retry. Similarly, for another error occurring because of failure of Schematron validation, you can perform the action human intervention. This fault can be recovered manually by editing the payload and then resubmitting through Oracle Enterprise Manager.

Conditions are defined in the `fault-policies.xml` file, as shown in the following example:

```
<Conditions>
  <faultName xmlns:medns="http://schemas.oracle.com/mediator/faults"
name="medns:mediatorFault">
    <condition>
```

```

        <test>contains($fault.mediatorErrorCode, "TYPE_DATA_
TRANSFORMATION")</test>
        <action ref="ora-java"/>
    </condition>
</faultName>
<faultName xmlns:medns="http://schemas.oracle.com/mediator/faults"
name="medns:mediatorFault">
    <condition>
        <test>contains($fault.mediatorErrorCode, "TYPE_FATAL_MESH")</test>
        <action ref="ora-retry"/>
    </condition>
</faultName>
<faultName xmlns:medns="http://schemas.oracle.com/mediator/faults"
name="medns:mediatorFault">
    <condition>
        <test>contains($fault.mediatorErrorCode, "TYPE_DATA_ASSIGN")</test>
        <action ref="ora-retry-crm-endpoint"/>
    </condition>
</faultName>
</Conditions>

```

Identifying Fault Types Using Conditions

You can categorize the faults that can be captured using conditions in the following types:

- Mediator-specific faults

For all Mediator-specific faults, Mediator engine throws only one fault, namely {http://schemas.oracle.com/mediator/faults}mediatorFault. Every Mediator fault is wrapped into this fault. The errors or faults generated by a Mediator composite can be captured by using the following format:

```

<faultName xmlns:medns="http://schemas.oracle.com/mediator/faults"
name="medns:mediatorFault">
<!-- mediatorFault is a bucket for all the mediator faults -->
    <condition>
        <test>
            contains($fault.mediatorErrorCode, "TYPE_FATAL_MESH")
        </test>
<!-- Captures TYPE_FATAL_MESH errors -->
        <action ref="ora-retry"/>
    </condition>
</faultName>

```

- Business faults and SOAP faults

The errors or faults that can be captured by defining an XPath condition, which is based on the fault payload. For example:

```

<faultName xmlns:ns1="http://xmlns.oracle.com/Customer"
name="ns1:InvalidCustomer"> <!-- QName of Business/SOAP fault -->
    <condition>
        <test>
            contains($fault.<PART_NAME>/custid, 1011)
        </test>
<!-- xpath condition based on fault payload -->
        <action ref="ora-retry"/>
    </condition>
</faultName>

```

When a reference service returns a business fault, the fault can be handled in the Mediator component. The returned fault can be forwarded to another component, redirected to an adapter service like File adapter, or an event can be raised. But, if both fault policy and fault handler are defined for a business fault, then fault policy takes precedence over fault handler. In such a case, the fault handlers in the Mediator component are ignored, if the fault policy is successfully executed.

- Adapter-specific fault

The errors or faults generated by an Adapter can be captured by using the following format:

```
<faultName xmlns:medns="http://schemas.oracle.com/mediator/faults"
name="medns:mediatorFault">
  <condition>
    <test>${fault.faultCode} = "1"</test> <!-- unique constraint violation in DB
adapter-->
    <action ref="ora-retry"/>
  </condition>
</faultName>
```

20.1.1.2 Actions

Actions specify the tasks that should be performed when an error occurs. Mediator provides a list of actions that you can use in a fault policy. These predefined actions are described in the following list:

- Retry: Retry actions like enqueueing a message to a JMS queue that is stopped, or inserting a record with unique key constraint error and so on, enable you to retry a task that caused the error. A new thread is started with every retry action. So, with every retry action, a new transaction starts. The options available with retry action are:

Option	Description
Retry Count	Retry N times
Retry Interval	Delay in seconds for retry
Exponential Backoff	Retry interval increase with exponential backoff
Retry Failure Action	Chain to this action if retry N times fails
Retry Success Action	Chain to this action if retry succeeds

Note: Exponential backoff indicates that the next retry attempt is scheduled at $2 \times$ the *delay*, where *delay* is the current retry interval. For example, if the current retry interval is 2 seconds, the next retry attempt is scheduled at 4, the next at 8, and the next at 16 seconds until the `retryCount` value is reached.

The following code snippet shows how to specify the Retry action:

```
<Action id="ora-retry">
```

```

<retry>
<retryCount>3</retryCount>
<retryInterval>2</retryInterval>
<exponentialBackoff/>
<retryFailureAction ref="ora-java"/>
<retrySuccessAction ref="ora-java"/>
</retry>
</Action>

```

If you set the Retry Interval in the fault policy to a duration less than 30 seconds, then the retry may not happen within the specified intervals. This is because the default value of the `org.quartz.scheduler.idleWaitTime` property is 30 seconds, and the scheduler waits for 30 seconds before retrying for available triggers, when the scheduler is otherwise idle. If the Retry Interval is set to a value less than 30 seconds, then latency is expected.

If you want the system to use a retry interval that is less than 30 seconds, then add the following property under the section `<property name="quartzProperties">` in the `fabric-config-core.xml` file:

```
org.quartz.scheduler.idleWaitTime=<value>
```

- Human intervention: You can specify this action in the following way:

```
<Action id="ora-human-intervention"><humanIntervention/></Action>
```

- Abort: This action enables you to abort the flow. You can specify this action in the following way:

```
<Action id="ora-terminate"><abort/></Action>
```

- Java code: This action enables you to call a customized Java class that implements the `oracle.integration.platform.faultpolicy.IFaultRecoveryJavaClass` interface. You can specify this action in the following way:

Note: The implemented Java class must implement a method that returns a `String`. The policy can be chained to a new action based on the returned `String`.

```

<Action id="ora-java">
  <javaAction className="mypackage.myClass"
defaultAction="ora-terminate">
    <returnValue value="ABORT" ref="ora-terminate"/>
    <returnValue value="RETRY" ref="ora-retry"/>
    <returnValue value="MANUAL" ref="ora-human-intervention"/>
  </javaAction>
</Action>

```

For more information about

`oracle.integration.platform.faultpolicy.IFaultRecoveryJavaClass` interface and

`oracle.integration.platform.faultpolicy.IFaultRecoveryContext` interface, see the SOA Javadoc.

20.1.2 Fault Bindings

Fault bindings associate fault policies with composites or components, and are defined in the `fault-bindings.xml` file. The `fault-bindings.xml` file should be created based on the XML schema defined in [Section 20.4.2, "Schema Definition File for Fault-bindings.xml"](#).

Fault policies can be created at the following levels:

- **Composite:** You can define one fault policy for all Mediator components in a composite. You can specify this level in the following way:

```
<composite faultPolicy="ConnectionFaults"/>
```

- **Component:** You can define a fault policy for a Mediator component exclusively. A component-level fault policy overrides the composite-level fault policy. You can specify this level in the following way:

```
<component faultPolicy="ConnectionFaults">
  <name>Component1</name>
  <name>Component2</name>
</component>
```

- **Reference:** You can define a fault policy for the references of a Mediator component. You can specify this level in the following way:

```
<reference faultPolicy="policy1">
  <name>DBAdapter3</name>
</reference>
```

Note: Human intervention is the default action for errors that do not have a fault policy defined.

A sample fault binding file is shown in the following example:

```
<?xml version="1.0" encoding="UTF-8"?>
<faultPolicyBindings version="2.0.1"
  xmlns="http://schemas.oracle.com/bpel/faultpolicy"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <composite faultPolicy="ConnectionFaults"/>
</faultPolicyBindings>
```

20.1.3 Error groups in Mediator

You can specify an action for an error type or error group while defining the conditions in a fault policy. In the previous examples, `medns:mediatorFault` refers that the error is a Mediator error, whereas `medns:TYPE_FATAL_MESH` refers to an error group. An error group consists of one or more child error types. `TYPE_ALL` is an error group that contains all Mediator errors.

The following list describes various error groups contained in the `TYPE_ALL` error group:

- `TYPE_DATA`: Contains errors related to data handling.
 - `TYPE_DATA_ASSIGN`: Contains errors related to data assignment.
 - `TYPE_DATA_FILTERING`: Contains errors related to data filtering.

- TYPE_DATA_TRANSFORMATION: Contains errors that occur during transformation.
- TYPE_DATA_VALIDATION: Contains errors that occur during payload validation.
- TYPE_METADATA: Contains errors related to Mediator metadata.
 - TYPE_METADATA_FILTERING: Contains errors that occur while processing the filtering conditions.
 - TYPE_METADATA_TRANSFORMATION: Contains errors that occur during getting the metadata for transformation.
 - TYPE_METADATA_VALIDATION: Contains errors that occur during validation of metadata for Mediator (.mplan file).
 - TYPE_METADATA_COMMON: Contains other errors that occur during the handling of metadata.
- TYPE_FATAL: Contains fatal errors that are not easily recoverable.
 - TYPE_FATAL_DB: Contains database related fatal errors, such as Datasource not found error.
 - TYPE_FATAL_CACHE: Contains Mediator cache-related fatal errors.
 - TYPE_FATAL_ERRORHANDLING: Contains fatal errors that occur during error handling such as Resubmission queues not available.
 - TYPE_FATAL_MESH: Contains fatal errors from the Service Infrastructure such as Invoke service not available.
 - TYPE_FATAL_MESSAGING: Contains fatal messaging errors arising from the Service Infrastructure.
 - TYPE_FATAL_TRANSACTION: Contains fatal errors related to transactions such as Commit can't be called on a transaction which is marked for rollback.
 - TYPE_FATAL_TRANSFORMATION: Contains fatal transformation errors such as error occurring because of the XPath functions used in a transformation.
- TYPE_TRANSIENT: Contains transient errors that can be recovered on retrying.
 - TYPE_TRANSIENT_MESH: Contains errors related to the Service Infrastructure.
 - TYPE_TRANSIENT_MESSAGING: Contains errors related to JMS such as enqueue, dequeue.
- TYPE_INTERNAL: Contains internal errors.

20.2 Using Error Handling with Mediator

You can enable error handling for a Mediator by using the `fault-policies.xml` and `fault-bindings.xml` files.

20.2.1 How to Use Error Handling for a Mediator Component

To enable error handling for a Mediator component:

1. Create a `fault-policies.xml` file based on the schema defined in the [Section 20.4.1, "Schema Definition File for Fault-policies.xml"](#).

2. Create a `fault-bindings.xml` file based on the schema defined in the [Section 20.4.2, "Schema Definition File for Fault-bindings.xml"](#).
3. Copy the `fault-policies.xml` and the `fault-bindings.xml` file to your SOA Composite project directory.
4. Deploy the SOA Composite project.

20.2.2 What Happens at Runtime

All the fault policies for a composite are loaded when the first error occurs. At runtime, Mediator checks whether there is any policy defined for the current error. If a fault policy is defined, then Mediator performs the action according to the configuration done in the fault policies file. If there is no fault policy defined, then the default action of human intervention is performed.

20.3 Fault Recovery Using Enterprise Manager

Apart from policy-based recovery using the fault policy file, you can also perform fault recovery actions on Mediator faults identified as recoverable in Oracle Enterprise Manager Fusion Middleware Control Console. This can be performed in the following ways:

- Manual recovery by modifying the payload using Enterprise Manager
- Bulk recovery without modifying the payload using Enterprise Manager
- Aborting a faulted instance using Enterprise Manager, if the user does not want to do any more processing on the instance.

For more information about fault recovery using Enterprise Manager, see *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite*.

20.4 Error Handling XML Schema Definition Files

This section describes the schema files for the `fault-policies.xml` and `fault-bindings.xml` files and consists of the following sections:

- [Section 20.4.1, "Schema Definition File for Fault-policies.xml"](#)
- [Section 20.4.2, "Schema Definition File for Fault-bindings.xml"](#)

20.4.1 Schema Definition File for Fault-policies.xml

The `fault-policies.xml` file should be based on the following XSD file:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="http://schemas.oracle.com/bpel/faultpolicy"
xmlns:tns="http://schemas.oracle.com/bpel/faultpolicy"
xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
  <!-- Conditions contain a list of fault names -->
  <xs:element name="Conditions">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="faultName" type="tns:faultNameType"
maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <!-- action Ref must exist in the same file -->
```

```

<xs:complexType name="actionRefType">
  <xs:attribute name="ref" type="xs:string" use="required"/>
</xs:complexType>
<!-- one condition has a test and action, if test is missing, this is the
catch all condition -->
<xs:complexType name="conditionType">
  <xs:all>
    <xs:element name="test" type="tns:idType" minOccurs="0"/>
    <xs:element name="action" type="tns:actionRefType"/>
  </xs:all>
</xs:complexType>
<!-- One fault name match contains several conditions -->
<xs:complexType name="faultNameType">
  <xs:sequence>
    <xs:element name="condition" type="tns:conditionType"
maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="name" type="xs:QName"/>
</xs:complexType>
<xs:complexType name="ActionType">
  <xs:choice>
    <xs:element name="retry" type="tns:RetryType"/>
    <xs:element ref="tns:rethrowFault"/>
    <xs:element ref="tns:humanIntervention"/>
    <xs:element ref="tns:abort"/>
    <xs:element ref="tns:replayScope"/>
    <xs:element name="javaAction" type="tns:JavaActionType">
      <xs:key name="UniqueReturnValue">
        <xs:selector xpath="tns:returnValue"/>
        <xs:field xpath="@value"/>
      </xs:key>
    </xs:element>
  </xs:choice>
  <xs:attribute name="id" type="tns:idType" use="required"/>
</xs:complexType>
<xs:element name="Actions">
  <xs:annotation>
    <xs:documentation>Fault Recovery Actions</xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Action" type="tns:ActionType"
maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:complexType name="JavaActionType">
  <xs:annotation>
    <xs:documentation>This action invokes java code
provided</xs:documentation>
  </xs:annotation>
  <xs:sequence>
    <xs:element name="returnValue" type="tns:ReturnValueType"
minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="className" type="tns:idType" use="required"/>
  <xs:attribute name="defaultAction" type="tns:idType" use="required"/>
  <xs:attribute name="propertySet" type="tns:idType"/>
</xs:complexType>
<xs:complexType name="RetryType">

```

```

        <xs:annotation>
            <xs:documentation>This action attempts retry of activity
execution</xs:documentation>
        </xs:annotation>
        <xs:all>
            <xs:element ref="tns:retryCount"/>
            <xs:element ref="tns:retryInterval"/>
            <xs:element ref="tns:exponentialBackoff" minOccurs="0"/>
            <xs:element name="retryFailureAction"
type="tns:retryFailureActionType" minOccurs="0"/>
            <xs:element name="retrySuccessAction"
type="tns:retrySuccessActionType" minOccurs="0"/>
        </xs:all>
    </xs:complexType>
    <xs:simpleType name="idType">
        <xs:restriction base="xs:string">
            <xs:minLength value="1"/>
        </xs:restriction>
    </xs:simpleType>
    <xs:complexType name="ReturnValueType">
        <xs:annotation>
            <xs:documentation>Return value from java code can chain another action
using
                return values</xs:documentation>
        </xs:annotation>
        <xs:attribute name="value" type="tns:idType" use="required"/>
        <xs:attribute name="ref" type="xs:string" use="required"/>
    </xs:complexType>
    <xs:element name="exponentialBackoff">
        <xs:annotation>
            <xs:documentation>Setting this will cause retry attempts to use
                exponentialBackoff algorithm</xs:documentation>
        </xs:annotation>
    </xs:complexType/>
</xs:element>
<xs:element name="humanIntervention">
    <xs:annotation>
        <xs:documentation>This action causes the activity to
freeze</xs:documentation>
    </xs:annotation>
</xs:complexType/>
</xs:element>
<xs:element name="replayScope">
    <xs:annotation>
        <xs:documentation>This action replays the immediate enclosing
scope</xs:documentation>
    </xs:annotation>
</xs:complexType/>
</xs:element>
<xs:element name="rethrowFault">
    <xs:annotation>
        <xs:documentation>This action will rethrow the
fault</xs:documentation>
    </xs:annotation>
</xs:complexType/>
</xs:element>
<xs:element name="retryCount" type="xs:positiveInteger">
    <xs:annotation>
        <xs:documentation>This value is used to identify number of
retries</xs:documentation>

```

```

        </xs:annotation>
    </xs:element>
    <xs:complexType name="retryFailureActionType">
        <xs:annotation>
            <xs:documentation>This is the action to be chained if retry attempts
fail</xs:documentation>
        </xs:annotation>
        <xs:attribute name="ref" type="xs:string" use="required"/>
    </xs:complexType>
    <xs:complexType name="retrySuccessActionType">
        <xs:annotation>
            <xs:documentation>This is the action to be chained if retry attempts
is successful</xs:documentation>
        </xs:annotation>
        <xs:attribute name="ref" type="xs:string" use="required"/>
    </xs:complexType>
    <xs:element name="retryInterval" type="xs:unsignedLong">
        <xs:annotation>
            <xs:documentation>This is the delay in milliseconds of retry
attempts</xs:documentation>
        </xs:annotation>
    </xs:element>
    <xs:element name="abort">
        <xs:annotation>
            <xs:documentation>This action terminates the
process</xs:documentation>
        </xs:annotation>
        <xs:complexType/>
    </xs:element>
    <xs:element name="Properties">
        <xs:annotation>
            <xs:documentation>Properties that can be passes to a custom java
class</xs:documentation>
        </xs:annotation>
        <xs:complexType>
            <xs:sequence>
                <xs:element name="propertySet" type="tns:PropertySetType"
maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:complexType name="PropertySetType">
        <xs:sequence>
            <xs:element name="property" type="tns:PropertyValueType"
maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attribute name="name" type="tns:idType" use="required"/>
    </xs:complexType>
    <xs:complexType name="PropertyValueType">
        <xs:simpleContent>
            <xs:extension base="tns:idType">
                <xs:attribute name="name" type="tns:idType" use="required"/>
            </xs:extension>
        </xs:simpleContent>
    </xs:complexType>
    <xs:element name="faultPolicy">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="tns:Conditions"/>
                <xs:element ref="tns:Actions"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>

```

```

        <xs:element ref="tns:Properties" minOccurs="0"/>
        <!--Every policy has on Conditions and and one Actions, however,
Properties is optional -->
    </xs:sequence>
    <xs:attribute name="id" type="tns:idType" use="required"/>
    <xs:attribute name="version" type="xs:string" default="2.0.1"/>
</xs:complexType>
<xs:key name="UniqueActionId">
    <xs:selector xpath="tns:Actions/tns:Action"/>
    <xs:field xpath="@id"/>
</xs:key>
<xs:key name="UniquePropertySetId">
    <xs:selector xpath="tns:Properties/tns:property_set"/>
    <xs:field xpath="@id"/>
</xs:key>
<xs:keyref name="RetryActionRef" refer="tns:UniqueActionId">
    <xs:selector
xpath="tns:Actions/tns:Action/tns:retry/tns:retryFailureAction"/>
    <xs:field xpath="@ref"/>
</xs:keyref>
<xs:keyref name="RetrySuccessActionRef" refer="tns:UniqueActionId">
    <xs:selector
xpath="tns:Actions/tns:Action/tns:retry/tns:retrySuccessAction"/>
    <xs:field xpath="@ref"/>
</xs:keyref>
<xs:keyref name="JavaActionRef" refer="tns:UniqueActionId">
    <xs:selector
xpath="tns:Actions/tns:Action/tns:javaAction/tns:returnValue"/>
    <xs:field xpath="@ref"/>
</xs:keyref>
<xs:keyref name="ConditionActionRef" refer="tns:UniqueActionId">
    <xs:selector
xpath="tns:Conditions/tns:faultName/tns:condition/tns:action"/>
    <xs:field xpath="@ref"/>
</xs:keyref>
<xs:keyref name="JavaDefaultActionRef" refer="tns:UniqueActionId">
    <xs:selector xpath="tns:Actions/tns:Action/tns:javaAction"/>
    <xs:field xpath="@defaultAction"/>
</xs:keyref>
<xs:keyref name="JavaPropertySetRef" refer="tns:UniquePropertySetId">
    <xs:selector xpath="tns:Actions/tns:Action/tns:javaAction"/>
    <xs:field xpath="@property_set"/>
</xs:keyref>
</xs:element>
<xs:element name="faultPolicies">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="tns:faultPolicy" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
</xs:schema>

```

20.4.2 Schema Definition File for Fault-bindings.xml

The `fault-bindings.xml` file should be based on the following XSD file:

```

<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="http://schemas.oracle.com/bpel/faultpolicy"
    xmlns:tns="http://schemas.oracle.com/bpel/faultpolicy"

```

```

xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
  <xs:element name="faultPolicyBindings">
    <xs:annotation>
      <xs:documentation>Bindings to a specific fault policy
    </xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element name="composite" type="tns:compositeType"
minOccurs="0" maxOccurs="1"/>
        <xs:element name="component" type="tns:componentType"
minOccurs="0" maxOccurs="unbounded"/>
        <xs:element name="reference" type="tns:referenceType"
minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="version" type="xs:string" default="2.0.1"/>
    </xs:complexType>
    <xs:key name="UniquePartnerLinkName">
      <xs:selector xpath="tns:reference/tns:name"/>
      <xs:field xpath="."/>
    </xs:key>
    <xs:key name="UniquePortType">
      <xs:selector xpath="tns:reference/tns:portType"/>
      <xs:field xpath="."/>
    </xs:key>
    <xs:key name="UniquePolicyName">
      <xs:selector xpath="tns:reference"/>
      <xs:field xpath="@faultPolicy"/>
    </xs:key>
  </xs:element>
  <xs:simpleType name="nameType">
    <xs:restriction base="xs:string">
      <xs:minLength value="1"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:complexType name="propertyType">
    <xs:simpleContent>
      <xs:extension base="tns:nameType">
        <xs:attribute name="name" type="xs:string" use="required"
fixed="faultPolicy"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>

  <xs:complexType name="referenceType">
    <xs:annotation>
      <xs:documentation>Bindings for a partner link. Overrides composite
level binding.</xs:documentation>
    </xs:annotation>
    <xs:sequence>
      <xs:annotation>
        <xs:documentation>Specification at partner link name overrides
specification for a port type</xs:documentation>
      </xs:annotation>
      <xs:element name="name" type="tns:nameType" minOccurs="0"
maxOccurs="unbounded"/>
      <xs:element name="portType" type="xs:QName" minOccurs="0"
maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="faultPolicy" type="tns:nameType" use="required"/>
  </xs:complexType>

```

```
</xs:complexType>

<xs:complexType name="componentType">
  <xs:annotation>
    <xs:documentation>Binding for a component </xs:documentation>
  </xs:annotation>
  <xs:sequence>
    <xs:element name="name" type="tns:nameType" minOccurs="0"
maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="faultPolicy" type="tns:nameType" use="required"/>
</xs:complexType>
<xs:complexType name="compositeType">
  <xs:annotation>
    <xs:documentation>Binding for the entire composite</xs:documentation>
  </xs:annotation>
  <xs:attribute name="faultPolicy" type="tns:nameType" use="required"/>
</xs:complexType>
</xs:schema>
```

Working with Multiple Part Messages in Mediator

This chapter describes how to use multiple part (multipart) messages with Oracle Mediator (Mediator) service component.

This chapter includes the following section:

- [Section 21.1, "Introduction to Mediator Multipart Message Support Feature"](#)

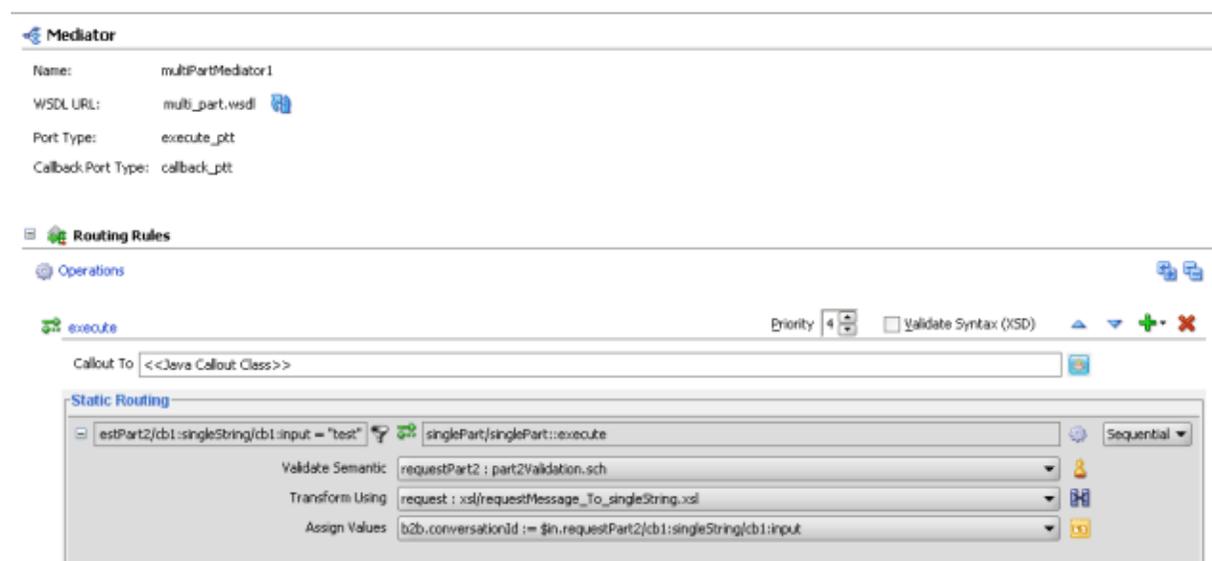
21.1 Introduction to Mediator Multipart Message Support Feature

Oracle Application Server 11g release includes support for importing multipart WSDL files in the JDeveloper Mediator Editor.

Oracle Mediator supports working with multipart source and target messages, which include multipart filter expression building, multipart schema validation, and transformations between multipart source and target messages for requests, replies, faults, and callbacks.

The Mediator Editor for a multipart source looks like [Figure 21-1](#).

Figure 21-1 Mediator Editor for a Multipart Source



This section covers the following sections:

- [Section 21.1.1.1, "Working with Multipart Request Messages"](#)
- [Section 21.1.2, "Working with Multipart Reply, Fault, and Callback Source Messages"](#)
- [Section 21.1.3, "Working with Multipart Target Messages"](#)

21.1.1 Working with Multipart Request Messages

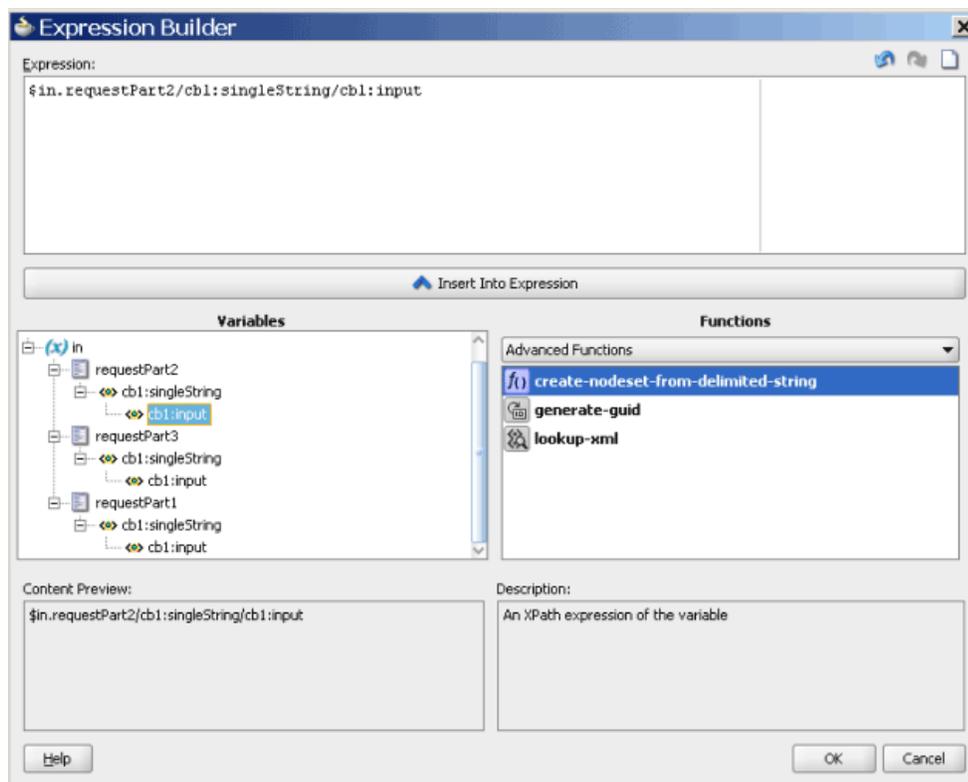
This section deals with multipart request messages. It contains the following sections:

- [Section 21.1.1.1, "Specifying Filter Expressions"](#)
- [Section 21.1.1.2, "Adding Validations"](#)
- [Section 21.1.1.3, "Creating Transformations"](#)
- [Section 21.1.1.4, "Assigning Values"](#)

21.1.1.1 Specifying Filter Expressions

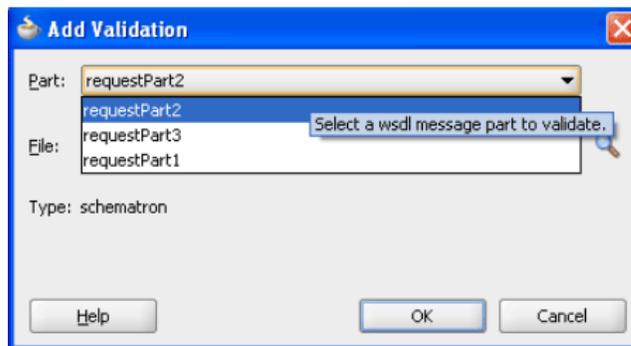
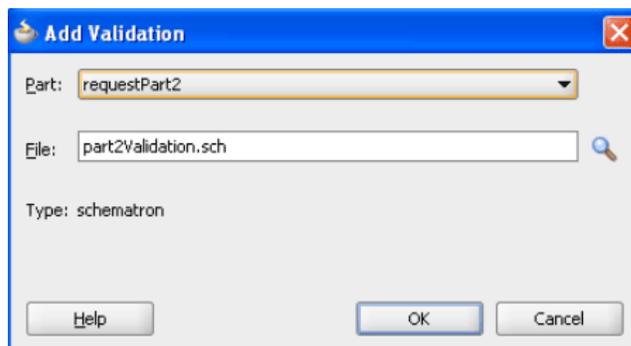
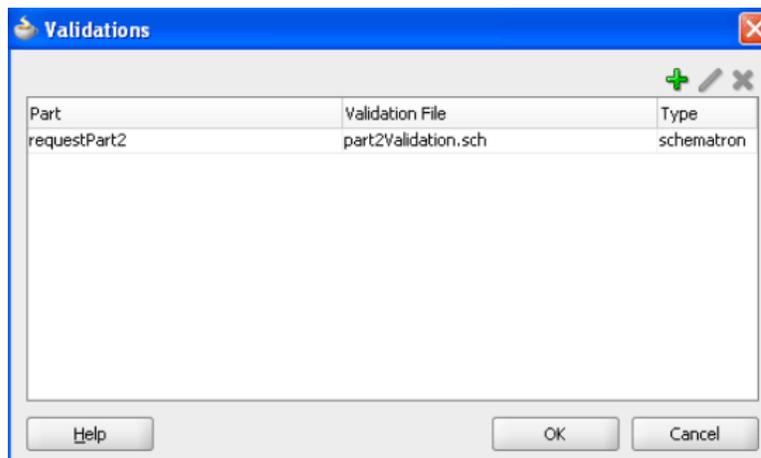
If you specify a filter expression for a multipart message, then the expression builder displays all message parts under the `in` variable, as shown in [Figure 21–2](#):

Figure 21–2 Expression Builder for a Multipart Request Source



21.1.1.2 Adding Validations

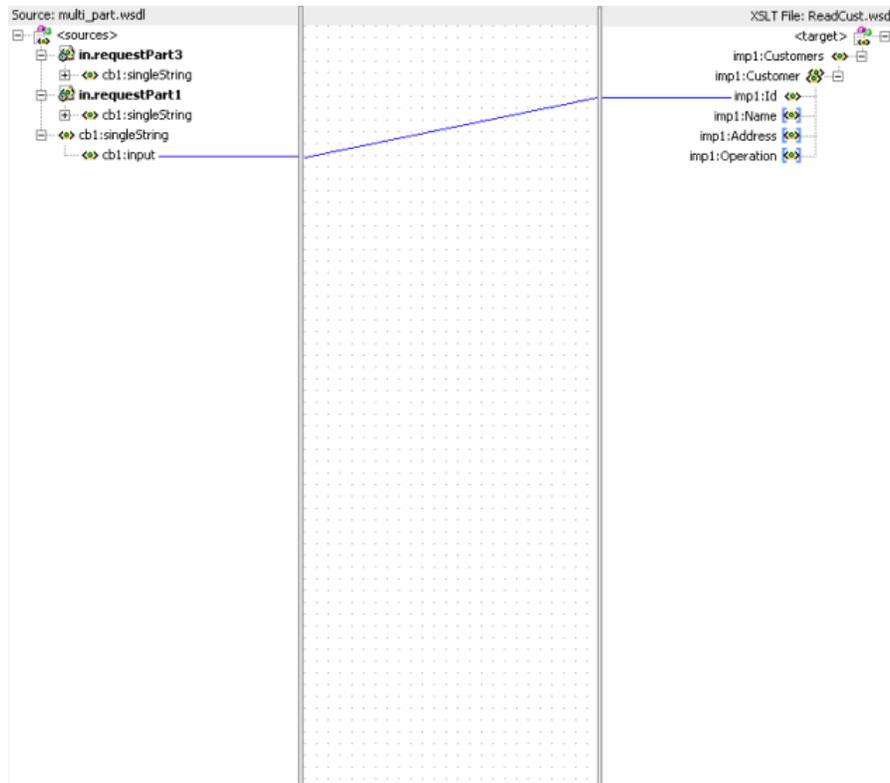
If you add a validation for a multipart message, then the Add Validation dialog displays a list of parts, from where you can choose one part. You specify a Schematron file for each request message part and then Mediator processes the Schematron files for the parts. This is shown in [Figure 21–3](#), [Figure 21–4](#), and [Figure 21–5](#):

Figure 21–3 Add Validation Dialog for a Multipart Request Source**Figure 21–4 Add Validation Dialog for a Multipart Request Source****Figure 21–5 Validations Dialog for a Multipart Request Source**

21.1.1.3 Creating Transformations

If you create a new mapper file for a multipart message, then the generated mapper file shows multiple source parts in the XSLT Mapper transformation tool as shown in [Figure 21–6](#):

Figure 21–6 XSLT Mapper transformation tool for a Multipart Request Source



21.1.1.4 Assigning Values

If you assign values using a source expression, then the expression builder shows an in variable for each message part. This is the same as specifying filter expressions. [Figure 21–7](#), [Figure 21–8](#), and [Figure 21–9](#) illustrate how you can assign values to a multipart message.

Figure 21–7 Assign Values Dialog for a Multipart Request Source

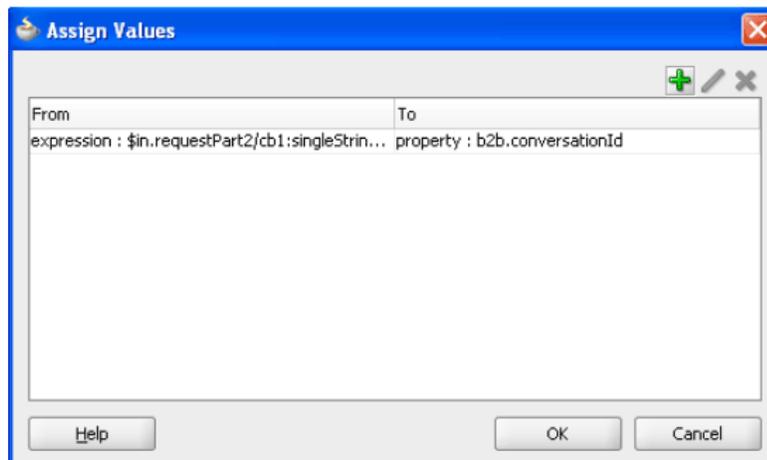
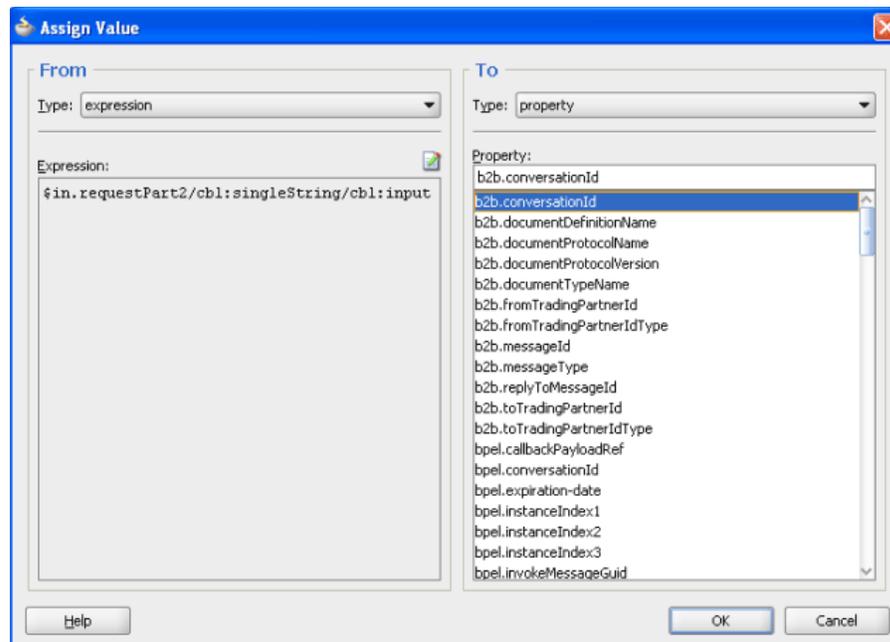
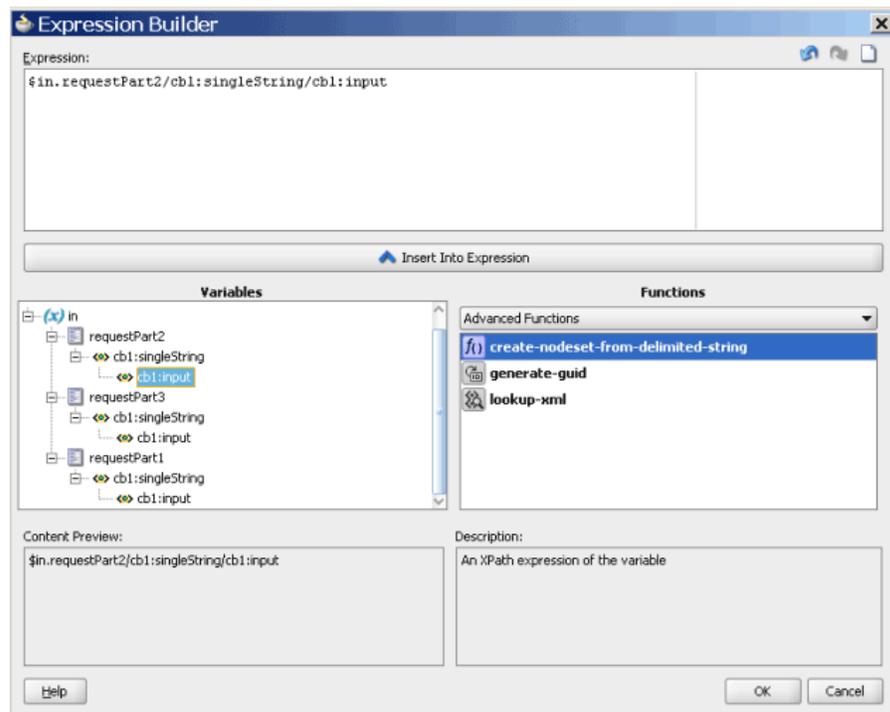


Figure 21–8 Assign Value Dialog for a Multipart Request Source**Figure 21–9 Expression Builder for a Multipart Request Source**

21.1.2 Working with Multipart Reply, Fault, and Callback Source Messages

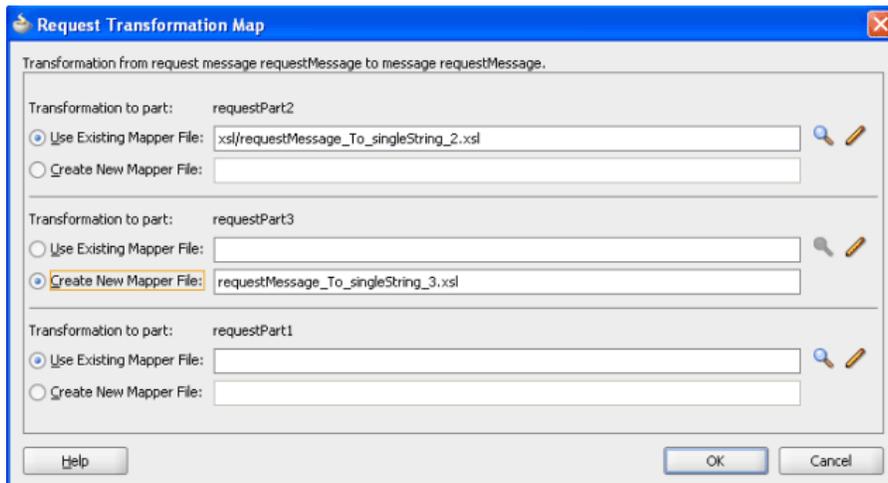
The method to create transformations and assigning values to multipart Reply, Fault, and Callback source messages, is the same as working with request source message.

Note: You cannot specify filter expressions or add validations for Reply, Fault, and Callback messages.

21.1.3 Working with Multipart Target Messages

If a routing target, that is, a request, reply, fault, or callback has a multipart message, then the transformation is handled in a slightly different way. This is because the XSLT Mapper transformation tool does not support multipart targets. So, in such a case, the Mediator creates and coordinates a separate mapper file for each target part as shown in [Figure 21–10](#):

Figure 21–10 Request Transformation Map for a Multipart Routing Target



Understanding Message Exchange Patterns of a Mediator

This chapter describes common message exchange patterns between an Oracle Mediator (Mediator) component and other applications.

This chapter includes the following sections:

- Section 22.1, "Understanding One-way Message Exchange Pattern"
- Section 22.2, "Understanding Request-Reply Message Exchange Pattern"
- Section 22.3, "Understanding Request-Reply-Fault Message Exchange Pattern"
- Section 22.4, "Understanding Request-Callback Message Exchange Pattern"
- Section 22.5, "Understanding Request-Reply-Callback Message Exchange Pattern"
- Section 22.6, "Understanding Request-Reply-Fault-Callback Message Exchange Pattern"

Note: The following exchange patterns show the default handling of responses, faults, and callbacks by JDeveloper, when a routing rule is created. Keep in mind the following points for all the cases:

- When a response, fault, or callback is sent back to the caller, it is also possible to route the same to a different target service or event by clicking the button next to the target and selecting a different target.
 - When the caller of the Mediator expects a response, one or more routing rules may route the request to a target that does not return a response, but there should be *at least one* sequential routing rule that returns a response.
 - If you have multiple routing rules having request-response interaction, then the routing rules that send the response back to the initial caller, should precede other routing rules, if any, that forward the response.
 - The asynchronous request-reply pattern in Mediator is supported only for Web Services. This exchange pattern is not supported for Adapters and other services.
-
-

22.1 Understanding One-way Message Exchange Pattern

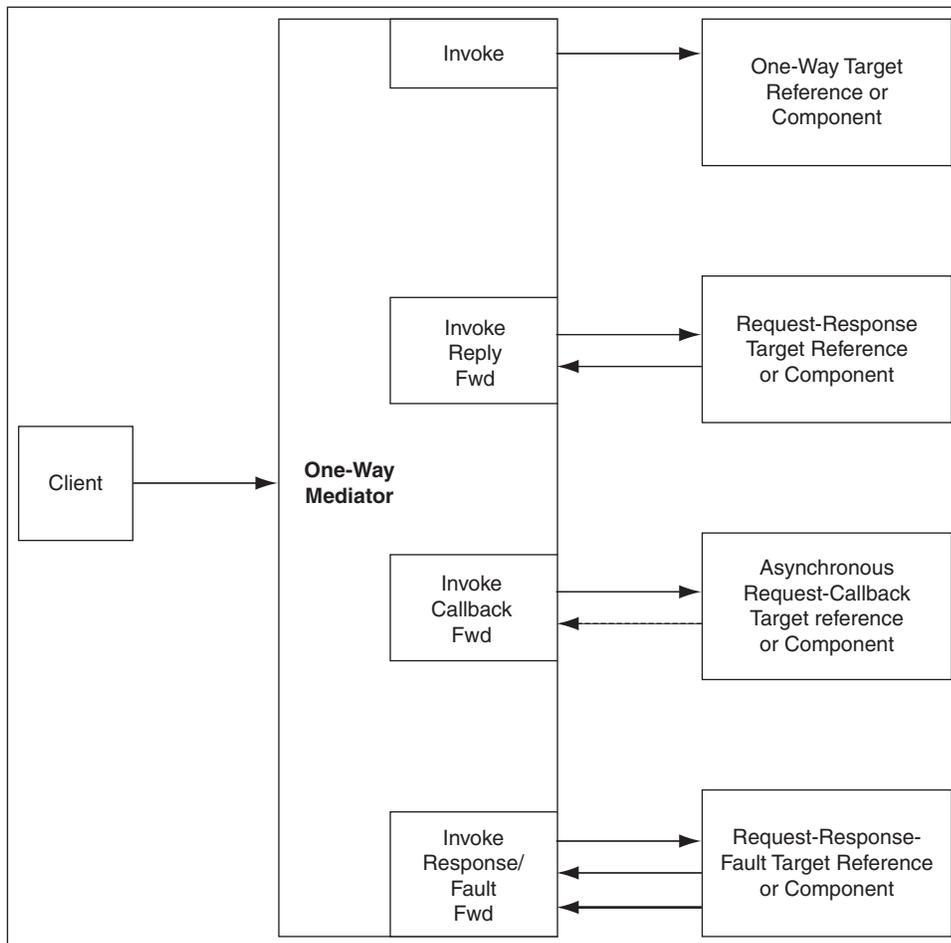
In a one-way interaction, the Mediator is invoked, but it does not send a response back to the caller. Depending on the type of routing rule target, the responses, faults, and callbacks are handled as shown in [Table 22-1](#):

Table 22-1 Response When Mediator's WSDL Is One-way

Routing Rule Target Type	Response
Request	No response.
Request Response	Response is forwarded to another target or event.
Request Response Fault	Response and fault are forwarded to another target or event.
Request Callback	Callback is forwarded to another target or event.
Request Response Callback	Response and callback are forwarded to another target or event.
Request Response Fault Callback	Response, fault, and callback are forwarded to another target or event.

[Figure 22-1](#) illustrates one-way message exchange pattern.

Figure 22-1 One-way Message Exchange Pattern



22.2 Understanding Request-Reply Message Exchange Pattern

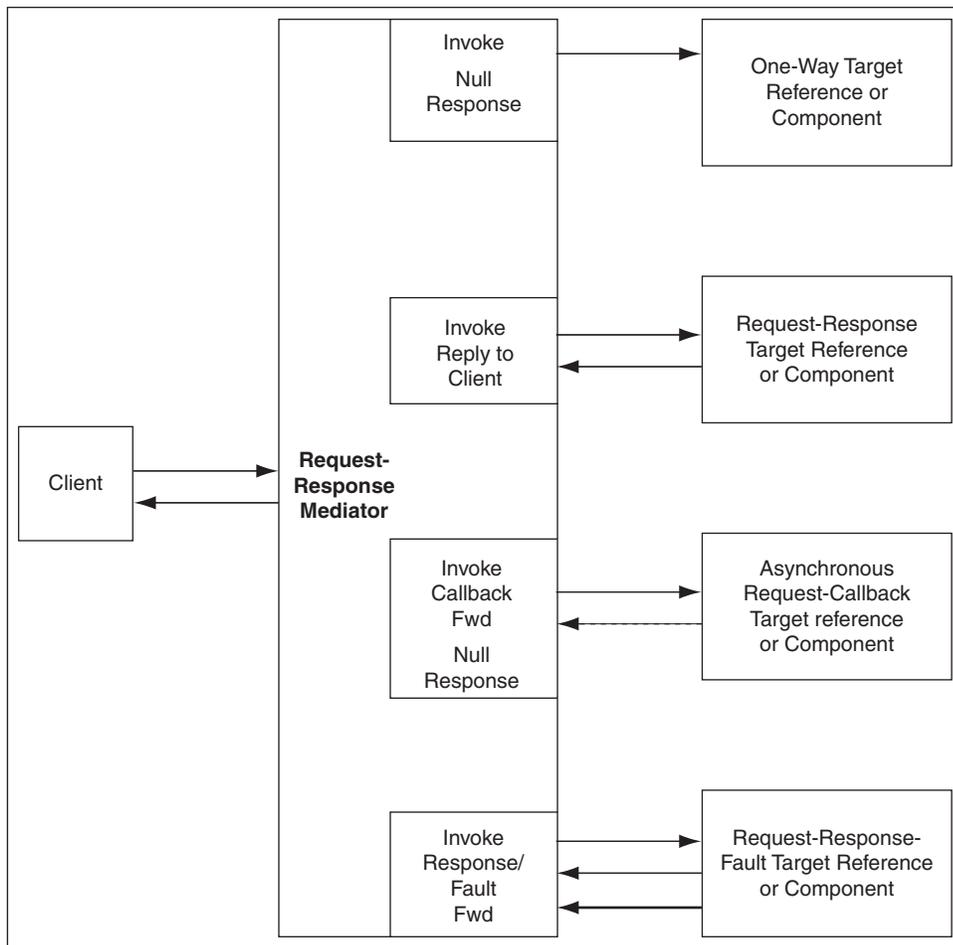
In a request-reply interaction, the Mediator is invoked, and the Mediator sends a reply to the caller. Depending on the type of routing rule target, the responses, faults, and callbacks are handled as shown in [Table 22-2](#):

Table 22-2 *Response When Mediator's WSDL Is Request Reply*

Routing Rule Target Type	Response
Request	No Response from the target, but there should be at least one sequential routing rule with request-response service.
Request Response	Response is sent back to the caller. Response can be forwarded to another target or event, but there should be at least one sequential routing rule that returns a response back to the caller.
Request Response Fault	Response is sent back to the caller. Fault is forwarded to another target or event.
Request Callback	No Response from the target, but there should be at least one sequential routing rule with request-response service. Callback is forwarded to another target or event.
Request Response Callback	Response is sent back to the caller. Callback is forwarded to another target or event.
Request Response Fault Callback	Response is sent back to the caller. Callback and fault are forwarded to another target or event.

[Figure 22-2](#) illustrates request-reply message exchange pattern.

Figure 22–2 Request-Reply Message Exchange Pattern



22.3 Understanding Request-Reply-Fault Message Exchange Pattern

In a request-reply-fault interaction, the Mediator is invoked and the Mediator sends a reply and one or more faults back to the caller. Depending on the type of routing rule target, the responses, faults, and callbacks are handled as shown in [Table 22–3](#):

Table 22–3 Response When Mediator’s WSDL Is Request Reply Fault

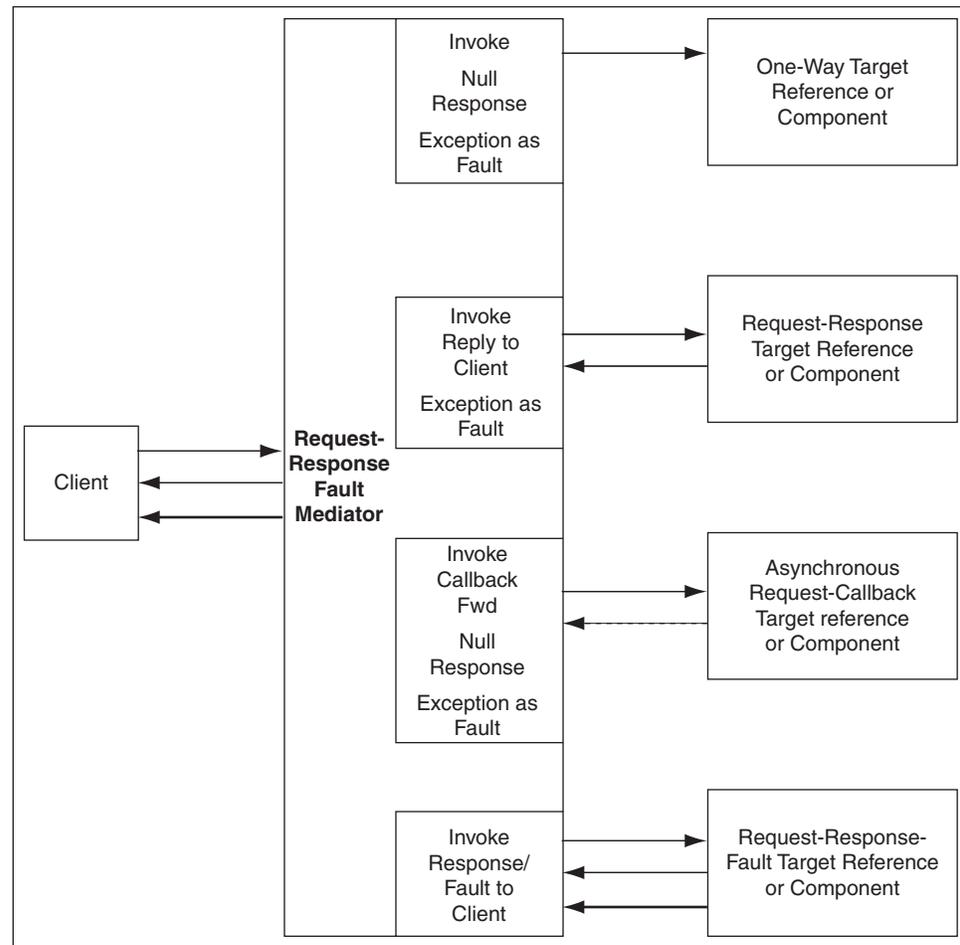
Routing Rule Target Type	Response
Request	There should be at least one sequential routing rule with request-response-fault service. Mediator returns null when there is no response to be sent.
Request Response	Response is sent back to the caller. Any exception in Mediator message processing may result in a fault.
Request Response Fault	Response and fault are sent back to the caller. Any exception in Mediator message processing may result in a fault.
Request Callback	No Response from the target, but there should be at least one sequential routing rule with request-response service. Mediator returns null when there is no response to be sent. Callback is forwarded to another target or event.
Request Response Callback	Response is sent back to the caller. Any exception in Mediator message processing may result in a fault.

Table 22-3 (Cont.) Response When Mediator's WSDL Is Request Reply Fault

Routing Rule Target Type	Response
Request Response Fault Callback	Response and fault are sent back to the caller. Any exception in Mediator message processing may result in a fault.

Figure 22-3 illustrates request-reply-fault message exchange pattern.

Figure 22-3 Request-Reply-Fault Message Exchange Pattern



22.4 Understanding Request-Callback Message Exchange Pattern

In a request-callback interaction, the Mediator is invoked and the Mediator may send an asynchronous reply to the caller. Depending on the type of routing rule target, the responses, faults, and callbacks are handled as shown in Table 22-4:

Table 22-4 Response When Mediator's WSDL Is Request Callback

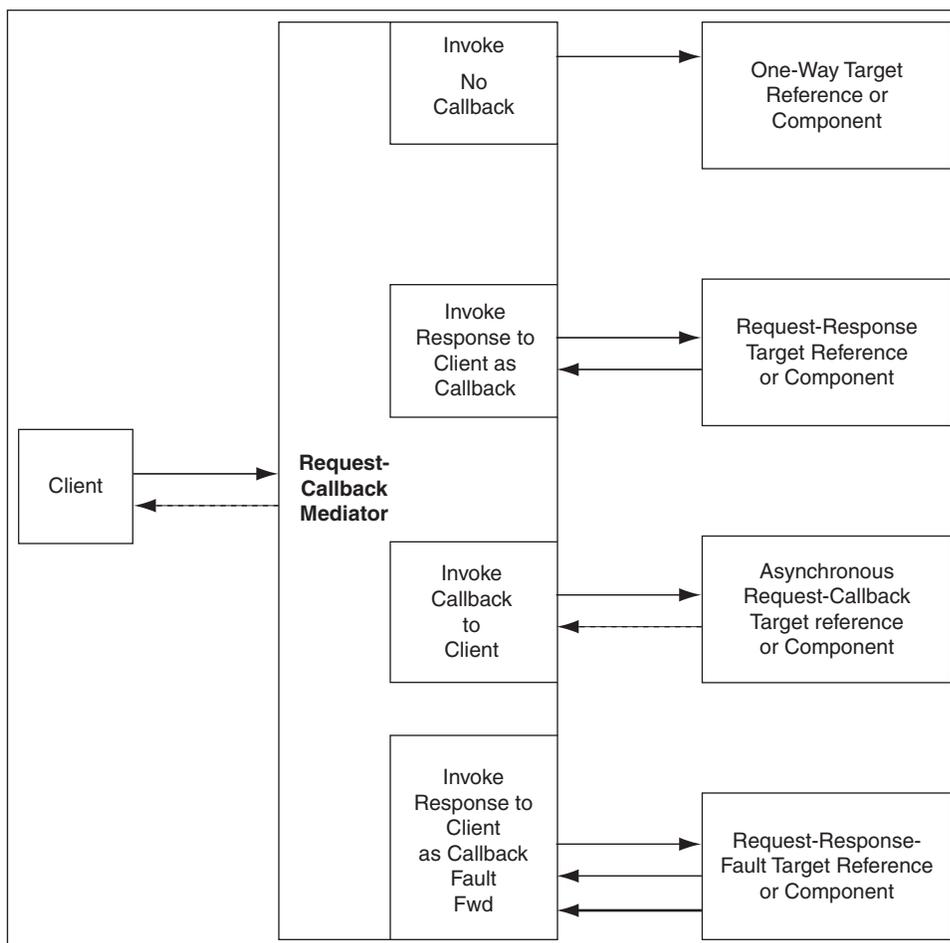
WSDL of the Routing Rule Target	Response
Request	There should be at least one sequential routing rule with request-callback service. No callback is sent to the caller if there is no routing rule with a defined callback.

Table 22–4 (Cont.) Response When Mediator's WSDL Is Request Callback

WSDL of the Routing Rule Target	Response
Request Response	Response is sent back to the caller, as callback, in a separate thread.
Request Response Fault	Response is sent back to the caller, as callback, in a separate thread. Fault is forwarded to another target or event.
Request Callback	Callback is sent back to the caller.
Request Response Callback	Callback is sent back to the caller, and response is forwarded to another target or event.
Request Response Fault Callback	Callback is sent back to the caller. Response and fault are forwarded to another target or event.

Figure 22–4 illustrates request-callback message exchange pattern.

Figure 22–4 Request-Callback Message Exchange Pattern



22.5 Understanding Request-Reply-Callback Message Exchange Pattern

In a request-reply-callback interaction, the Mediator is invoked and the Mediator sends a response and an asynchronous reply to the initial caller. Depending on the

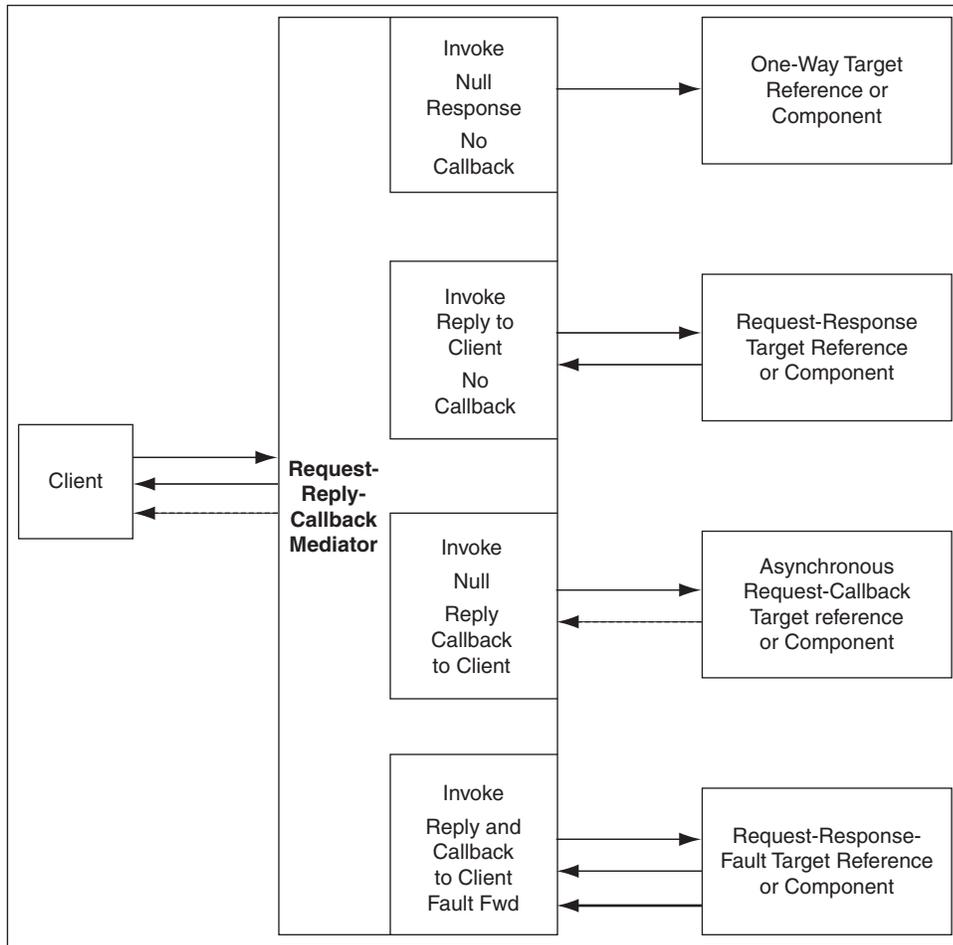
type of routing rule target, the responses, faults, and callbacks are handled as shown in [Table 22-5](#):

Table 22-5 Response When Mediator's WSDL Is Request Response Callback

Routing Rule Target Type	Response
Request	There should be at least one sequential routing rule that returns a response. No callback is sent to the caller if there is no routing rule with a defined callback.
Request Response	There should be at least one sequential routing rule that returns a response. No callback is sent, if there is no routing rule with a defined callback.
Request Response Fault	There should be at least one sequential routing rule that returns a response. No callback is sent to the caller if there is no routing rule with a defined callback.
Request Callback	There should be at least one sequential routing rule that returns a response. Mediator returns <code>null</code> when there is no response to be sent.
Request Response Callback	Response and callback are sent back to the caller.
Request Response Fault Callback	Response and callback are sent back to the caller. Fault is forwarded to another target or event.

[Figure 22-5](#) illustrates request-reply-callback message exchange pattern.

Figure 22–5 Request-Reply-Callback Message Exchange Pattern



22.6 Understanding Request-Reply-Fault-Callback Message Exchange Pattern

In a request-reply-fault-callback interaction, the Mediator is invoked and the Mediator sends a response, an asynchronous reply, and one or more fault types to the initial caller. Depending on the type of routing rule target, the responses, faults, and callbacks are handled as shown in [Table 22–6](#):

Table 22–6 Response to a Request Response Fault Callback Mediator

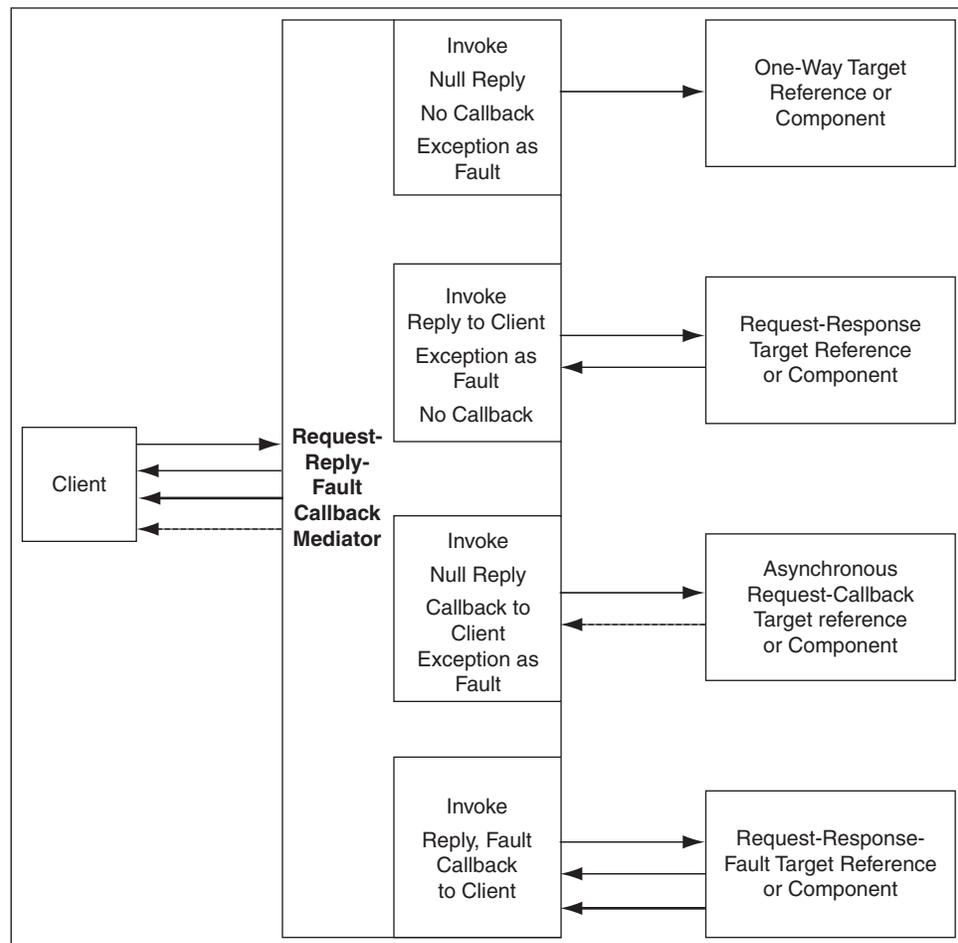
WSDL of the Routing Rule Target	Response
Request	There should be at least one sequential routing rule with request-callback service. No callback is sent to the caller if there is no routing rule with a defined callback.
Request Response	There should be at least one sequential routing rule with request-callback service. No callback is sent to the caller if there is no routing rule with a defined callback.
Request Response Fault	There should be at least one sequential routing rule with request-callback service. No callback is sent to the caller if there is no routing rule with a defined callback.

Table 22–6 (Cont.) Response to a Request Response Fault Callback Mediator

WSDL of the Routing Rule Target	Response
Request Callback	There should be at least one sequential routing rule that returns a response. Mediator returns null when there is no response to be sent.
Request Response Callback	Response and callback are sent back to the caller. Any exception in Mediator message processing may result in a fault.
Request Response Fault Callback	Response, fault, and callback are sent back to the caller.

Figure 22–6 illustrates request-reply-fault-callback message exchange pattern.

Figure 22–6 Request-Reply-Fault-Callback Message Exchange Pattern



Part IV

Using the Business Rules Service Component

This part describes how to use the business rules service component.

This part contains the following chapter:

- [Chapter 23, "Using the Business Rule Service Component"](#)

Using the Business Rule Service Component

This chapter describes how to use a business rule service component to integrate an SOA composite application with Oracle Business Rules. A business rule service component is also called a Decision component. You can add business rules as part of an SCA application or as part of a BPEL process.

This chapter includes the following sections:

- [Section 23.1, "Introduction to the Business Rule Service Component"](#)
- [Section 23.2, "Introduction to Creating and Editing Business Rules"](#)
- [Section 23.3, "Adding Business Rules to a BPEL Process"](#)
- [Section 23.4, "Adding Business Rules to an SOA Composite Application"](#)
- [Section 23.5, "Running Business Rules in a Composite Application"](#)

For more examples of using Oracle Business Rules, see:

- *Oracle Fusion Middleware User's Guide for Oracle Business Rules*
- *Oracle Fusion Middleware Tutorial for Running and Building an Application with Oracle SOA Suite*

23.1 Introduction to the Business Rule Service Component

A Decision component, also called a business rule service component, supports use of Oracle Business Rules in an SOA composite application. Decision components support the following SOA composite usage:

- A Decision component can be used within an SOA composite and wired to a BPEL component.
- A Decision component can be used within an SOA composite and used directly to run business rules.
- A Decision component can be used with the dynamic routing capability of Mediator.

For more information, see [Chapter 19, "Creating Mediator Routing Rules"](#).

- A Decision component can be used with the Advanced Routing Rules in Human Workflow.

For more information, see [Section 25.4, "Associating the Human Task Service Component with a BPEL Process"](#).

23.1.1 Integrating BPEL Processes, Business Rules, and Human Tasks

You can create an SOA composite application that includes BPEL process, business rule, and human task service components. These components are complementary technologies. BPEL processes focus on the orchestration of systems, services, and people. Business rules focus on decision making and policies. Human tasks enable you to model a workflow that describes the tasks for users or groups to perform as part of an end-to-end business process flow.

Some examples of where business rules can be used include:

- **Dynamic processing**

Rules can perform intelligent routing within the business process based on service level agreements or other guidelines. For example, if the customer requires a response within one day, send a loan application to the QuickLoan loan agency only. If the customer can wait longer, then route the request to three different loan agencies.
- **Externalize business rules in the process**

There are typically many conditions that must be evaluated as part of a business process. However, the parameters to these conditions can be changed independently of the process. For example, you provide loans only to customers with a credit score of at least 650. This value may be changed dynamically based on new guidelines set by business analysts.
- **Data validation and constraint checks**

Rules can validate input documents or apply additional constraints on requests. For example, a new customer request must always be accompanied with an employment verification letter and bank account details.
- **Human task routing**

Rules are frequently used in the context of human tasks in the business process:

 - Policy-based task assignments dispatch tasks to specific roles or users. For example, a process that handles incoming requests from a portal can route loan requests and insurance quotes to a different set of roles.
 - Load balancing of tasks among users. When a task is assigned to a set of users or a role, each user in that role acquires a set of tasks and acts on them in a specified time. For new incoming tasks, policies may be applied to set priorities on the task and put them in specific user queues. For example, a specific loan agent is assigned a maximum of 10 loans at any time.

For more information about creating business rules in the Human Task editor of a human task component, see [Section 25.3.7.2, "Specifying Advanced Task Routing Using Business Rules."](#)

23.2 Introduction to Creating and Editing Business Rules

This section describes how to get started with business rules and provides a brief introduction to the main sections of Oracle JDeveloper that you use to design business rules.

23.2.1 How to Create Business Rules Components

You can add Business Rule components using the SOA Composite Editor.

To create a Business Rule component:

1. Follow the instructions in [Table 23–1](#) to start Oracle JDeveloper.

Table 23–1 Starting Oracle JDeveloper

To Start...	On Windows...	On UNIX...
Oracle JDeveloper	Click <i>JDev_Oracle_</i> <i>Home\JDev\bin\jdev.exe</i> or create a shortcut	<code>\$ORACLE_HOME/jdev/bin/jdev</code>

2. Create a Business Rule service component through one of the following methods:

As a service component in an existing SOA composite application:

- a. From the Component Palette, drag a **Business Rule** service component into the SOA Composite Editor.

In a new application:

- a. From the Application Navigator, select **File > New > Applications > SOA Application**.

This starts the Create SOA Application wizard.

- b. In the Name your application page, enter an application name in the **Name** field.
- c. In the **Directory** field, enter a directory path in which to create the SOA composite application and project.
- d. Click **Next**.
- e. In the Name your project page, enter a unique project name in the **Project Name** field. The project name *must* be unique across SOA composite applications. This is because the uniqueness of a composite is determined by its project name. For example, do not perform the actions described in [Table 23–2](#).

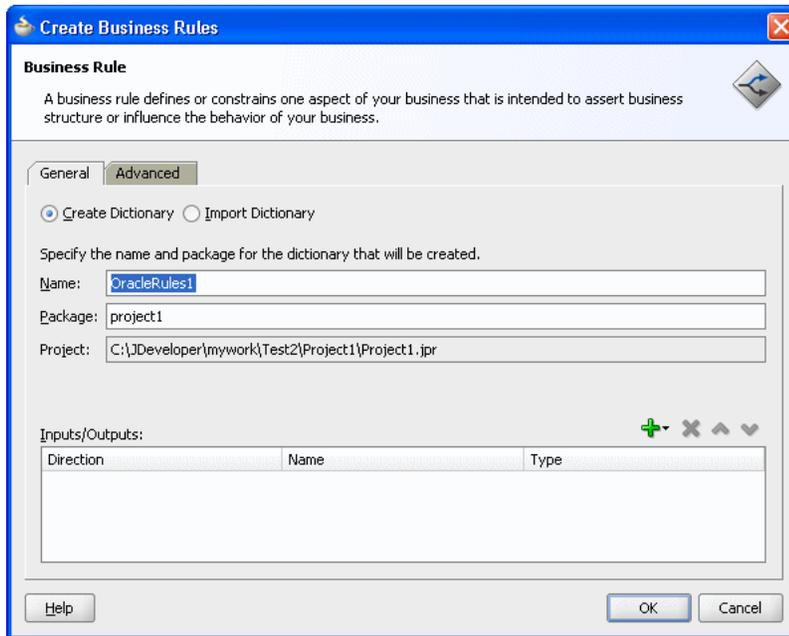
Table 23–2 Restrictions on Naming an SOA Project

Create an Application Named...	With an SOA Project Named...
Application1	Project1
Application2	Project1

During deployment, the second deployed project (composite) overwrites the first deployed project (composite).

- f. Click **Next**.
- g. In the Configure SOA settings page, select **Composite with Business Rule**.
- h. Click **Finish**.

Each method causes the Create Business Rules dialog shown in [Figure 23–1](#) to appear.

Figure 23–1 Create Business Rules Dialog

3. Provide the required details. For more information on providing Inputs and Outputs and on using the **Import Dictionary** option with this dialog, see *Oracle Fusion Middleware User's Guide for Oracle Business Rules*.
4. Click OK.

23.2.2 Introduction to Working with Business Rules in Rules Designer

When you are working with business rules Oracle JDeveloper displays Rules Designer. For more information on the windows shown with Rules Designer, see [Section 2.5, "Introduction to the Business Rules Designer"](#).

23.3 Adding Business Rules to a BPEL Process

You can use a Decision component, also called a business rule service component, to execute business rules in a BPEL process.

23.3.1 How to Add Business Rules to a BPEL Process

You add business rules to a BPEL process using a **Business Rule** component. When you add a business rule component to a BPEL process you need to include input and output variables to provide input to the rules and obtain results back from the business rules.

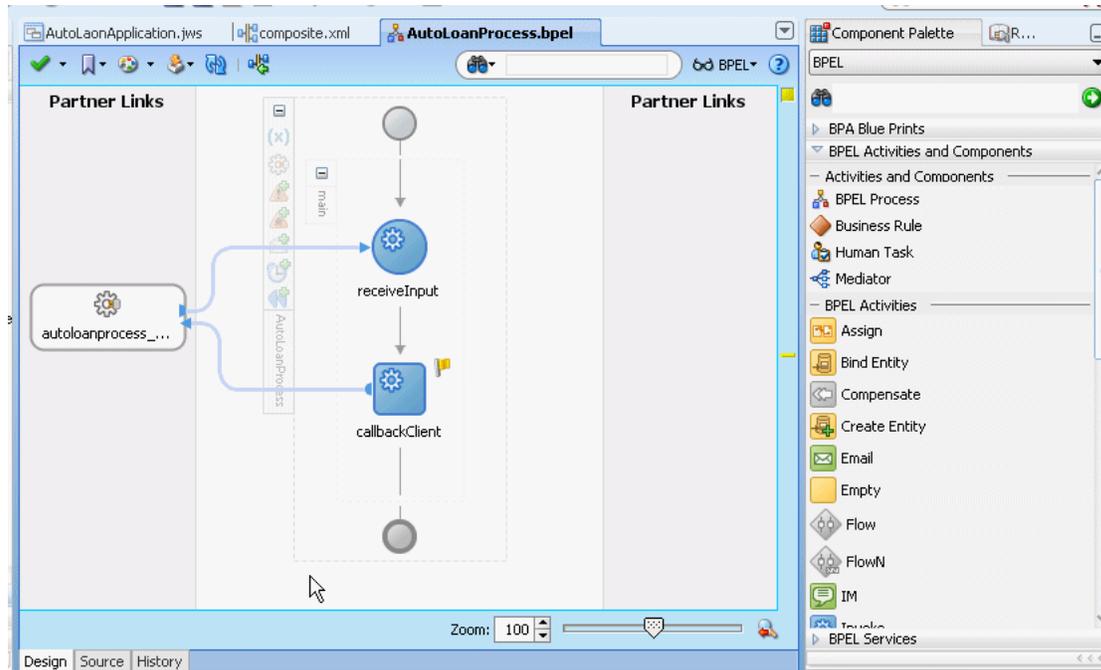
A business rule component enables you to execute business rules and make business decisions based on the rules. To create a business rule component, also called a Decision component, you drag-and-drop a **Business Rule** from the component palette into the BPEL process.

To add a business rule to a BPEL process:

1. Create a BPEL process service component. For more information, see [Section 5.1, "Introduction to the BPEL Process Service Component"](#).

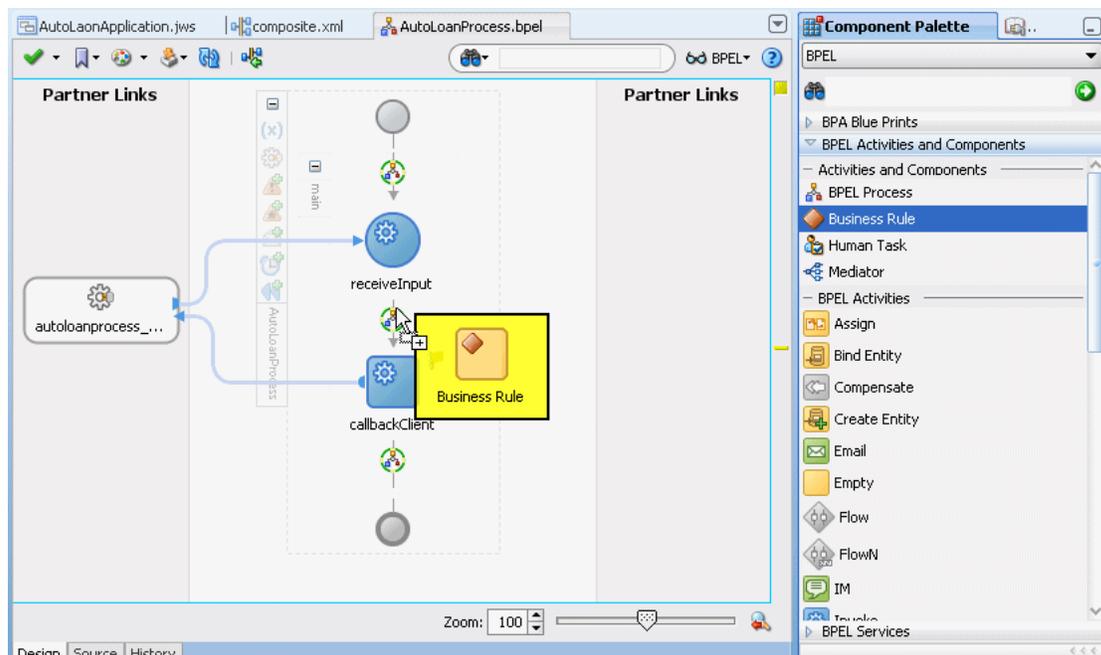
- Expand the BPEL process. For example, expand the BPEL process to view `receiveInput` and `callbackClient` as shown in [Figure 23-2](#).

Figure 23-2 Adding A Business Rule to a BPEL Process



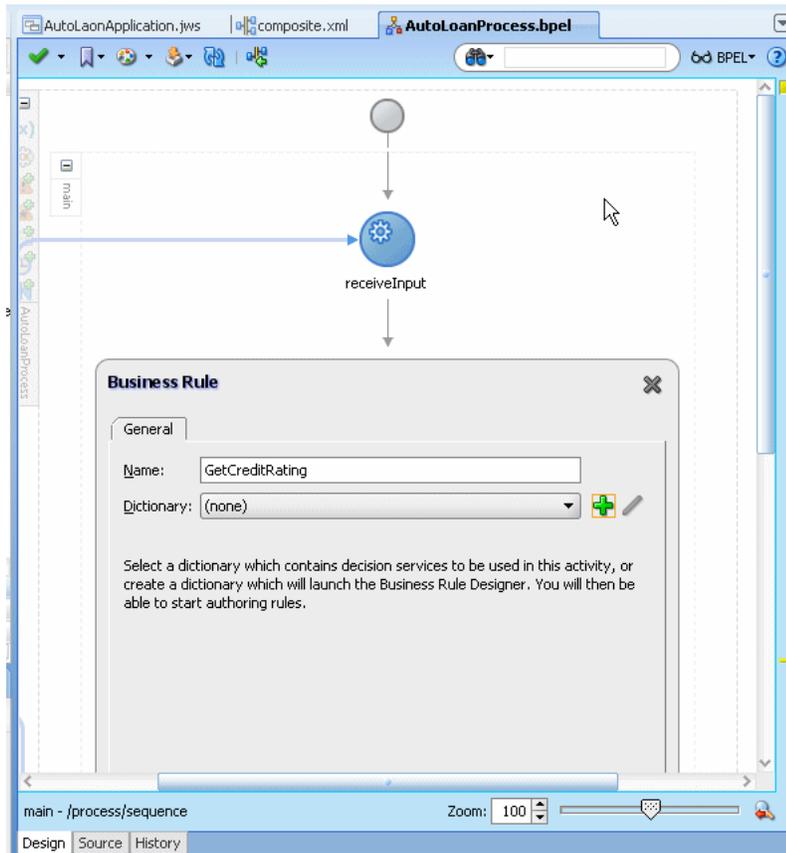
- Select **Business Rule** from the BPEL Activities and Components section of the Component Palette and drag-and-drop a **Business Rule** into the position where the business rules are needed. For example, drag-and-drop a **Business Rule** between `receiveInput` and `callbackClient`, as shown in [Figure 23-3](#).

Figure 23-3 Drag-and-drop a Business Rule into a BPEL Process



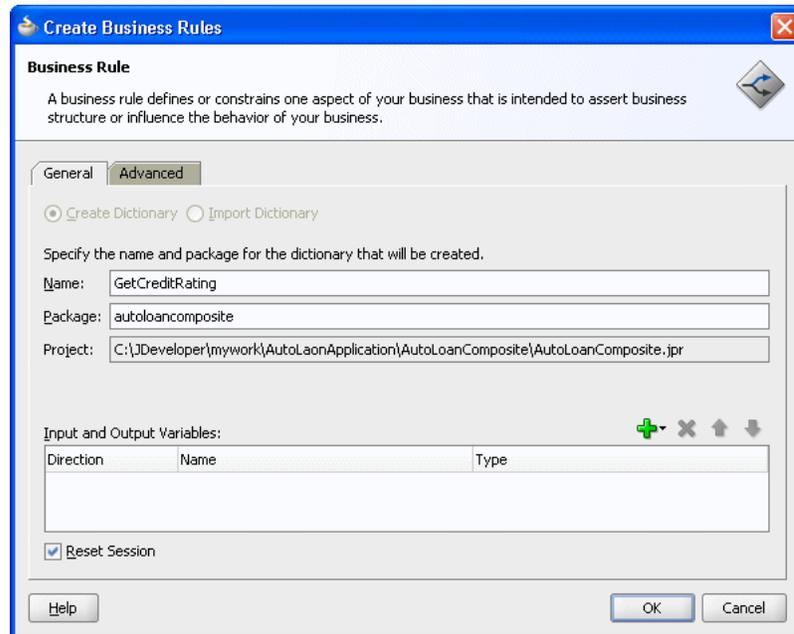
4. Oracle JDeveloper displays the business rule in the diagram. In the business rule area you can select an existing Oracle Business Rules dictionary or enter the name of a new dictionary to create. The Business Rule area includes a field to enter the business rule name. In the **Name** field enter a name. For example, enter `GetCreditRating`, as shown in [Figure 23-4](#). If you previously created a dictionary, in the **Dictionary** field, select an existing dictionary.

Figure 23-4 Business Rule Added to Auto Loan BPEL Process



5. In the Business Rule area for the **Business Rule Dictionary**, click the **Create Dictionary** icon to display the Create Business Rules dialog.
6. In the Create Business Rules dialog you do the following:
 - Specify a name for the Oracle Business Rules dictionary and a package name.
 - Specify the input and output data elements for the business rule. For example, for a sample Decision component named `GetCreditRating`, the input is a rating request document. The output is generated when you run the business rules, and for this example is a rating document. For example, in BPEL you can create two new variables, `RatingRequest` and `Rating` that carry the input and output data for the `GetCreditRating` rules.

Enter a name for the Oracle Business Rules dictionary. For example, enter `GetCreditRating`, as shown in [Figure 23-5](#).

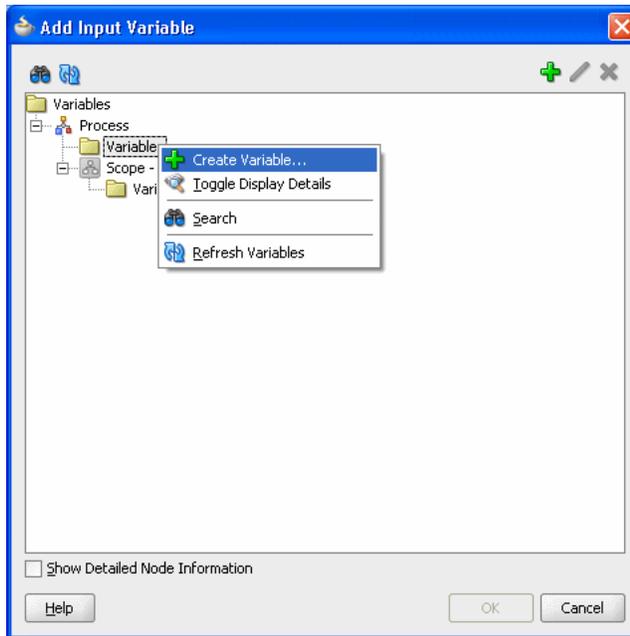
Figure 23–5 Adding GetCreditRating Business Rule Dictionary**Add inputs for business rule:**

1. In the Create Business Rules dialog, from the dropdown menu next to the **Add** icon select **Add Input Variable...** to create the input variable.

This displays the Add Input Variable dialog.

2. In the Add Input Variable dialog expand the **Process** folder and select the **Variables** folder immediately inside the **Process**.
3. Right-click the **Variables** folder and from the dropdown list select **Create Variable...** as shown in [Figure 23–6](#).

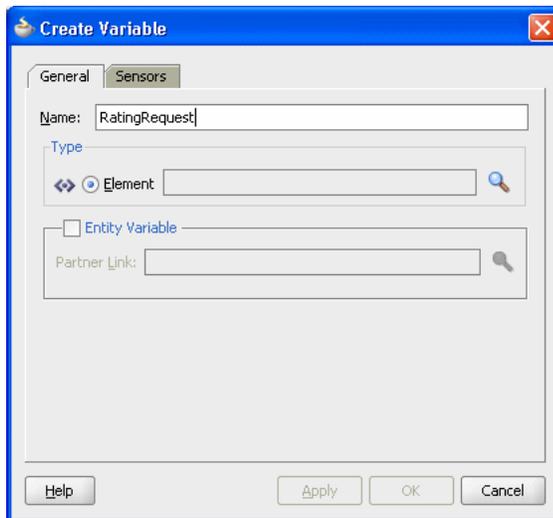
Figure 23–6 Add Input Variable



This displays the Create Variable dialog.

4. In the Create Variable dialog, in the **Name** field enter a value. For example, enter RatingRequest as shown in Figure 23–7.

Figure 23–7 Create Variable Dialog



5. In the Create Variable **Type** area click the **Browse Elements** icon. Use the navigator to locate the schema element type for the input variable. For example, select the ratingrequest type. Add any needed types using the Type Chooser.
6. Click the **Import Schema File** icon to import the schema. For example, import CreditRatingTypes.xsd. Also import any other required schema for your application.
7. In the Type Chooser dialog, select **ratingrequest** and click **OK**.

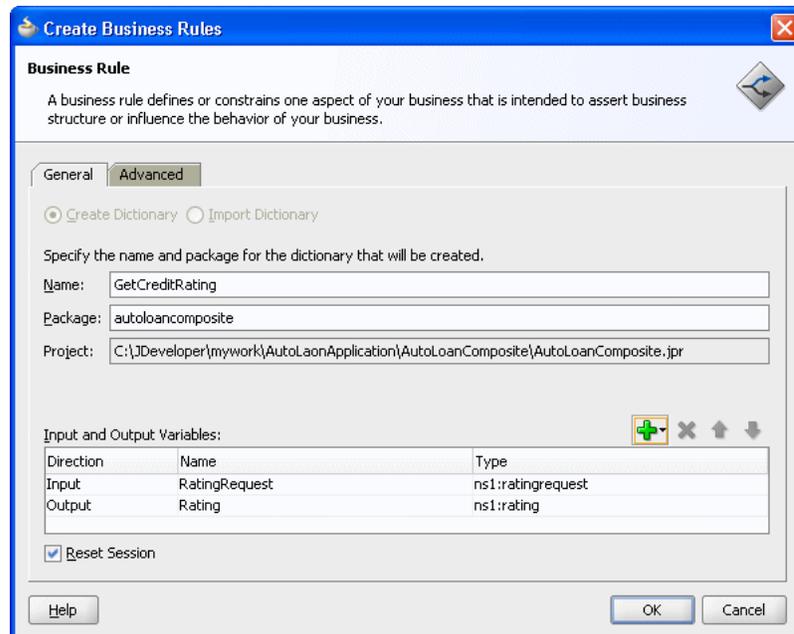
8. In the Create Variable dialog, click **OK**.
9. In the Add Input Variable dialog, click **OK**.

Add outputs for business rule:

1. In the Create Business Rules dialog, from the dropdown menu next to the **Add** icon, select **Add Output Variable....** This displays the Add Output Variable dialog. Use this dialog to create an output variable. For example, create an output variable for `GetCreditRating` in the same way you created the input variable.
2. In the Add Output Variable dialog select the scope by selecting the **Variables** folder under **Process**.
3. Right-click and from the dropdown list select **Create Variable....** This displays the Create Variable dialog.
4. In the Create Variable dialog, in the **Name** field enter the output variable name. For example enter `Rating`.
5. In the Create Variable dialog, in the **Type** area select the **Browse elements** icon and use the Type Chooser dialog to enter the type for the output variable. For example, expand the `CreditRatingTypes.xsd` and select the element type `rating`.
6. In the Type Chooser dialog, click **OK**.
7. In the Create Variable dialog, click **OK**.
8. In the Add Output Variable dialog, click **OK**.

This displays the Create Business Rules dialog, as shown in [Figure 23–8](#).

Figure 23–8 Create Business Rules Dialog with Input and Output Variables

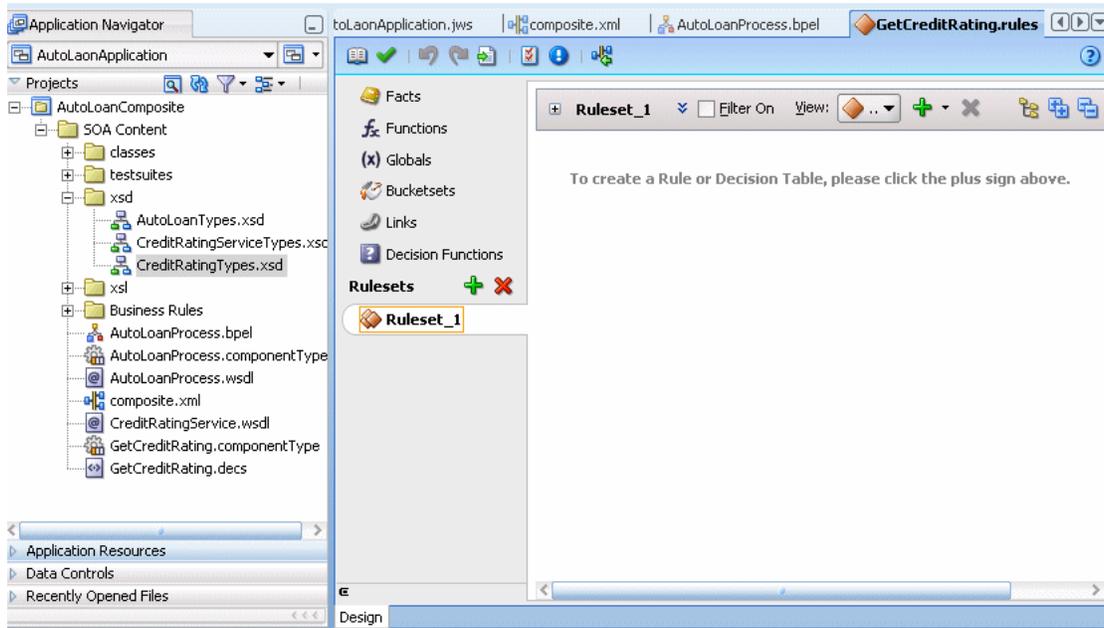


Set options and create decision service and business rules dictionary:

1. If you do not want to use the default service name, then select the **Advanced** tab and in the **Service Name** field enter the service name. For example enter the service name `CreditRatingService`.

2. Determine if the Decision Component is stateful or stateless with **Reset Session**. For more information, see [Section 23.3.5, "What You Need to Know About Decision Component Stateful Operation"](#).
3. In the Create Business Rules dialog, click **OK**. Oracle JDeveloper creates the Decision component and the dictionary and displays Rules Designer, as shown in [Figure 23–9](#).

Figure 23–9 Rules Designer Canvas Where You Work with Business Rules



For information on Rules Designer, see *Oracle Fusion Middleware User's Guide for Oracle Business Rules*.

23.3.2 What Happens When You Add Business Rules to a BPEL Process

When you add business rules to a BPEL process, Oracle JDeveloper creates a Decision component to control and run the business rules using the Business Rule Service Engine.

A Decision component consists of the following:

- Rules or Decision Tables that are evaluated using the Rules Engine. These are defined using Rules Designer and stored in a business rules dictionary.
- A description of the facts required for specific rules to be evaluated and the decision function to call. Each ruleset that contains rules or Decision Tables is exposed as a service with facts that are input and output, and the name of an Oracle Business Rules decision function. The facts are exposed through XSD definitions when you define the inputs and outputs for the business rule. A Decision function is stored in an Oracle Business Rules dictionary. For more information, see *Oracle Fusion Middleware User's Guide for Oracle Business Rules*.
- A web service wraps the input, output, and the call to the underlying Business Rule service engine.

This web service lets business processes assert and retract facts as part of the process. In some cases, all facts can be asserted from the business process as one

unit. In other cases, the business process can incrementally assert facts and eventually consult the rule engine for inferences. Therefore, the service supports both stateless and stateful interactions.

You can create a variety of such Decision components.

For more information, see *Oracle Fusion Middleware User's Guide for Oracle Business Rules*.

23.3.3 What Happens When You Create a Business Rules Dictionary

After you create an application, a project, and a rules dictionary, the rules dictionary appears in the structure pane in Oracle JDeveloper and Rules Designer opens in the main canvas.

As part of the create Business Rule dialog you either select an existing dictionary or a new rule dictionary is created with the following pre-loaded data:

- XML fact type model based on the input and output information of the Business Rule.
- A Ruleset that needs to be completed by adding rules or Decision Tables. With an existing dictionary, you use the import option to specify a dictionary that may already contain the rules or Decision Tables.
- A service component with the input and output contract of the Decision component.
- A Decision component for the rule dictionary and wires to the BPEL process.

23.3.4 What You Need to Know About Invoking Business Rules in a BPEL Process

When you add business rules to a BPEL process Oracle JDeveloper creates a Decision Service that supports calling Oracle Business Rules with the inputs you supply, and returning the outputs with results. The Decision Service provides access to Oracle Business Rules Engine at runtime as a web service. For more information, see *Oracle Fusion Middleware User's Guide for Oracle Business Rules*.

23.3.5 What You Need to Know About Decision Component Stateful Operation

A Decision Component running in a business rules service engine supports either stateful or stateless operation. The **Reset Session** checkbox in the Create Business Rules dialog provides support for these two modes of operation.

By default the **Reset Session** checkbox is selected which indicates stateless operation. Stateless operation means that, at runtime, the rule session is released after the Decision Component invocation.

When **Reset Session** is unselected, the underlying Oracle Business Rules object is kept in the memory of the business rules service engine at a separate location (so that it is not given back to the Rule Session Pool when the operation is finished). A subsequent use of the Decision component re-uses the cached RuleSession object, with all its state information from the `callFunctionStateful` invocation, and then releases it back to the Rule Session pool once the `callFunctionStateless` operation is finished. Thus, when **Reset Session** is unselected the rule session is saved for a subsequent request and a sequence of Decision Service invocations from the same BPEL process should always end with a stateless invocation.

23.4 Adding Business Rules to an SOA Composite Application

To work with Oracle Business Rules in an SOA composite application, you create an application and add business rules.

The business rule service component enables you to integrate your SOA composite application with business rules. This creates a business rule dictionary and enables you to execute business rules and make business decisions based on the rules.

After creating a project in Oracle JDeveloper, you need to create a Business Rule Service component within the project. When you add a business rule you can create input and output variables to provide input to the service component and to obtain results from the service component.

To use business rules with Oracle JDeveloper, you do the following:

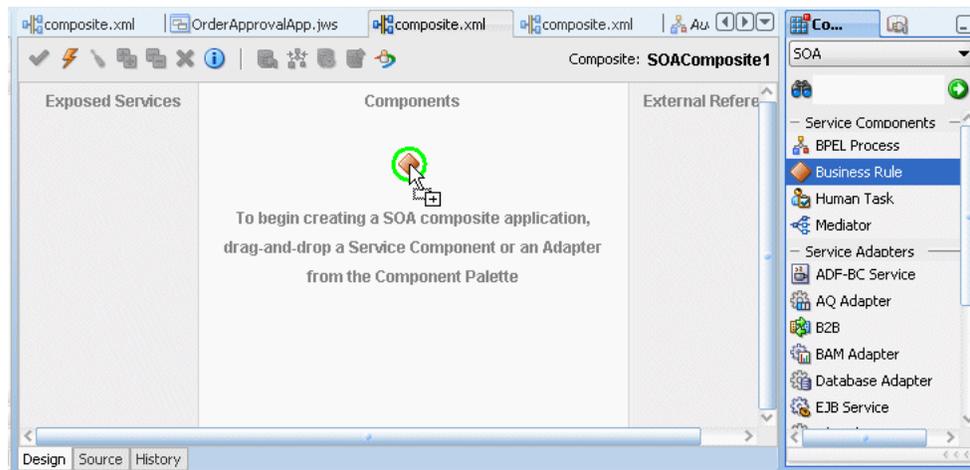
- Add a business rules service component
- Create input and output variables for the service component
- Create an Oracle Business Rules dictionary

23.4.1 How to Add Business Rules to an SOA Composite Application

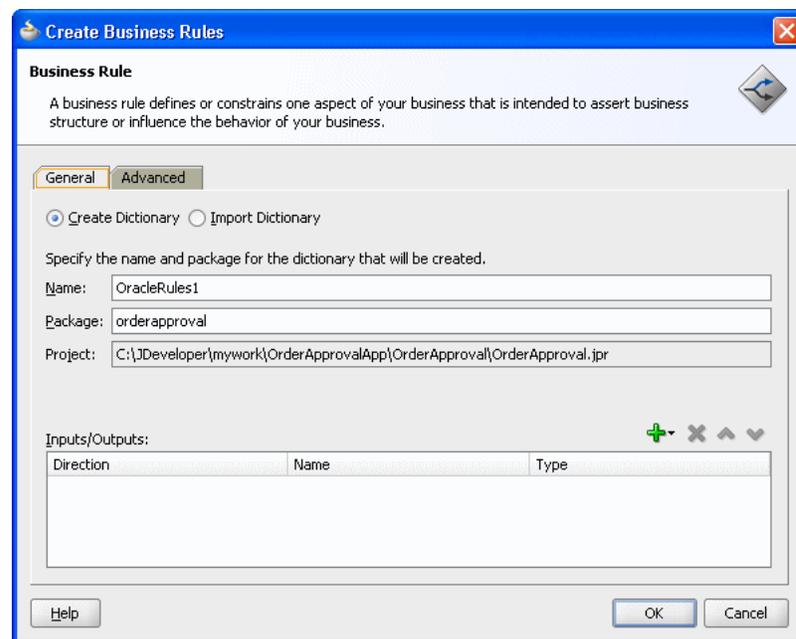
To work with Oracle Business Rules in an SOA composite application you use Oracle JDeveloper to create an application, a project, and then add a business rule component.

To create an SOA application with business rules:

1. Create an SOA application and project. For more information, see [Section 4.2.1, "How to Create an Application and a Project"](#). For an SOA composite using business rules, pick the required technologies for your application. For example, you may need the following for an SOA application with business rules: ADF Business Components, Java, and XML. You move these items to the **Selected** area on the Project Technologies tab.
2. In the Application Navigator, if the SOA composite editor is not showing, then in your project expand **SOA Content** folder and double-click `composite.xml` to launch the SOA composite editor.
3. From the Component Palette, drag-and-drop a **Business Rule** from the Service Components area of the SOA menu to the **Components** lane of the SOA composite editor, as shown in [Figure 23–10](#).

Figure 23–10 Adding Business Rules to an SOA Composite Application

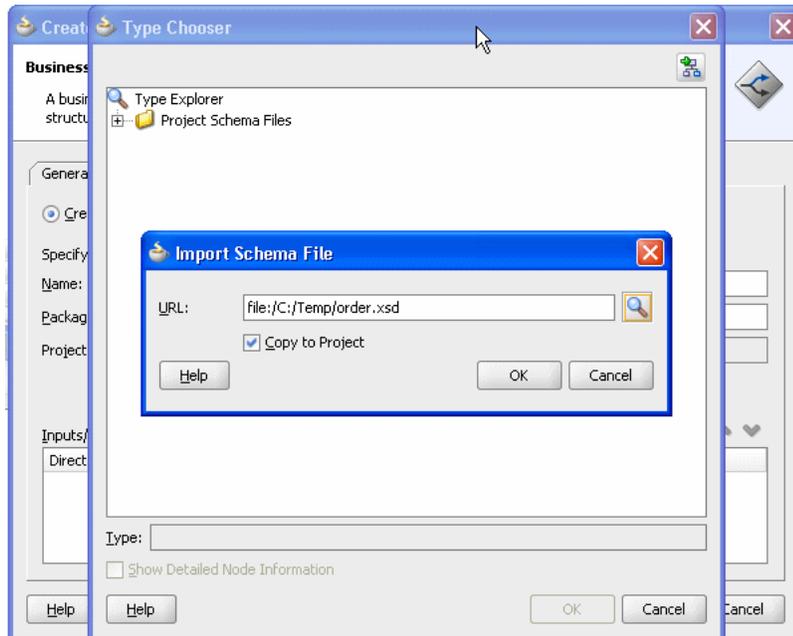
4. When you drag-and-drop a **Business Rule**, Oracle JDeveloper displays the Create Business Rules dialog as shown in Figure 23–11.

Figure 23–11 Adding Business Rules to an SOA Composite and Creating a Dictionary**Add inputs for business rules:**

1. In the Create Business Rules dialog, from the dropdown menu next to the **Add** icon select **Input...** to add input for the business rule. This displays the Type Chooser dialog.
2. In the Type Chooser dialog, add inputs. If the schema is available in the **Project Schema Files**, skip to step 9 to select the appropriate schema.
3. Click the **Import Schema File...** icon. This brings up the Import Schema File dialog.
4. In the Import Schema File dialog click **Browse Resources** to choose the XML schema elements for the input. This displays the SOA Resource Browser dialog.

5. In the SOA Resource Browser dialog, navigate to find the schema for your business rules input. For example, select the `order.xsd` schema file, and click **OK**.
6. In the Import Schema File dialog select **Copy to Project**, as shown in [Figure 23–12](#).

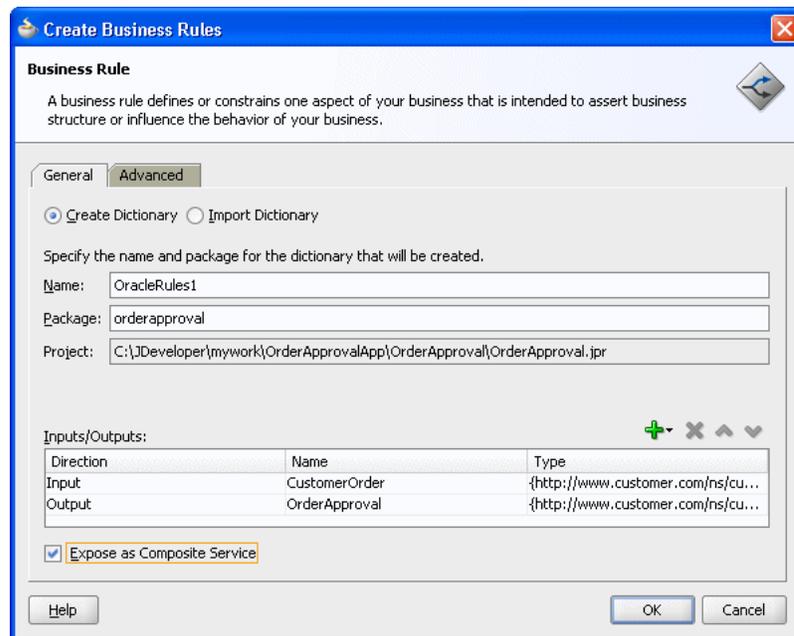
Figure 23–12 Importing Schema for Input to Business Rules



7. In the Import Schema File dialog, click **OK**.
8. In the Localize Files dialog, click **OK**.
9. Use the Type Chooser dialog navigator to locate and select the input from the schema and click **OK**. For example, select the `CustomerOrder` element as the input.

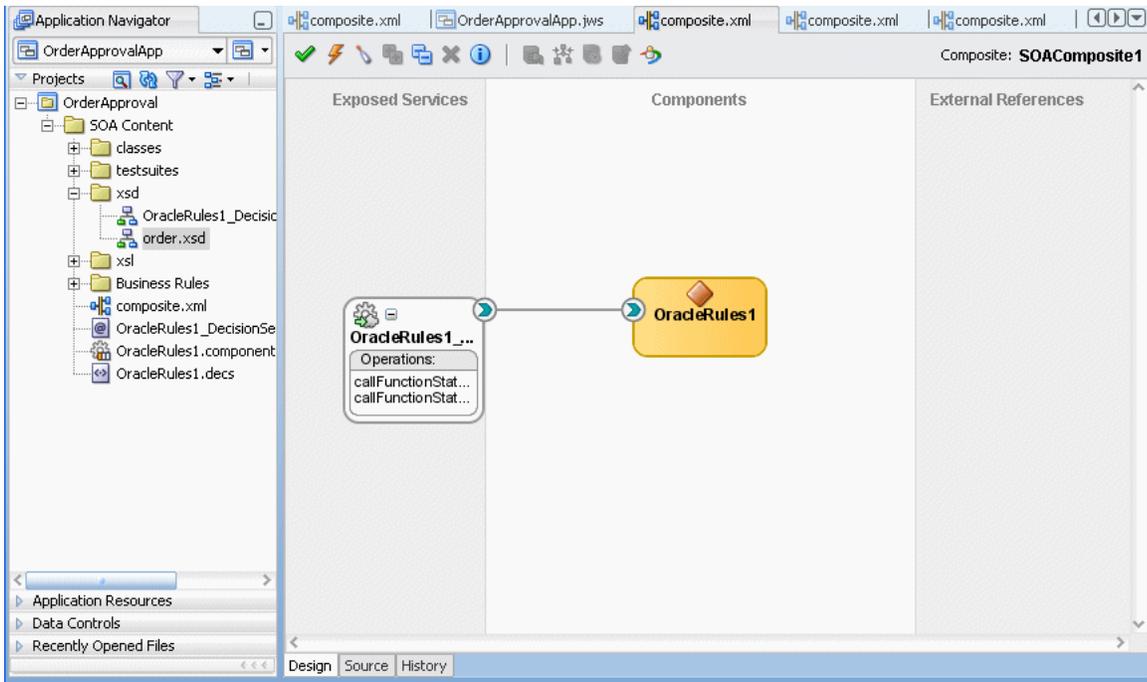
Add outputs for business rules:

1. In the Create Business Rules dialog, from the dropdown menu next to the **Add** icon select **Output...**
2. In the Type Chooser dialog, in a manner similar to adding an input add the output. For example, add `OrderApproval` from the `order.xsd` and click **OK**.
3. This displays the Create Business Rules dialog, as shown in [Figure 23–13](#).

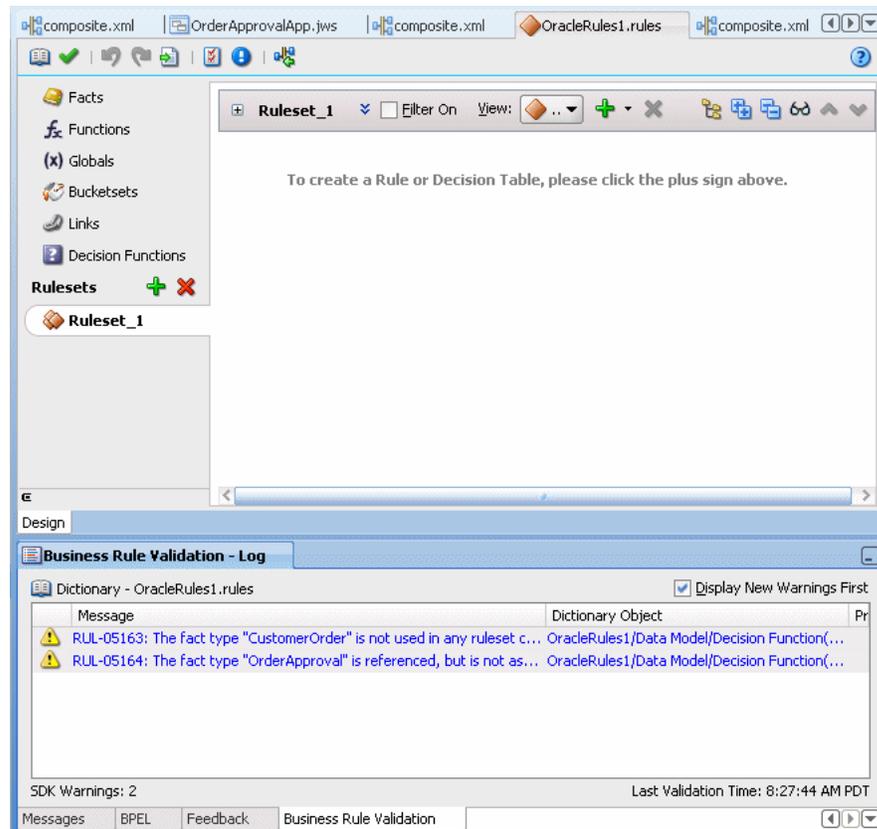
Figure 23–13 Create Business Rules Dialog with Input and Output**Set options and create decision service and business rules dictionary:**

1. In the Create Business Rules dialog, select **Expose as Composite Service**.
2. If you do not want to use the default service name, then select the **Advanced** tab and in the **Service Name** field enter the service name.
3. In the Create Business Rules dialog, click **OK**. This creates the Business Rule component, also called a Decision component, and Oracle JDeveloper shows the Business Rule component in the canvas workspace as shown in [Figure 23–14](#).

Figure 23–14 Business Rule Component in SOA Composite



4. Double-click the Decision component (for example the **OracleRules1** business rule). This opens Rules Designer, as shown in [Figure 23–15](#). The validation log shows validation warnings for the input and output facts. By working with Rules Designer to define rules or decision tables, you remove these warning messages.

Figure 23–15 Rules Designer Showing New Dictionary for SOA Composite Application

For information on Rules Designer, see *Oracle Fusion Middleware User's Guide for Oracle Business Rules*.

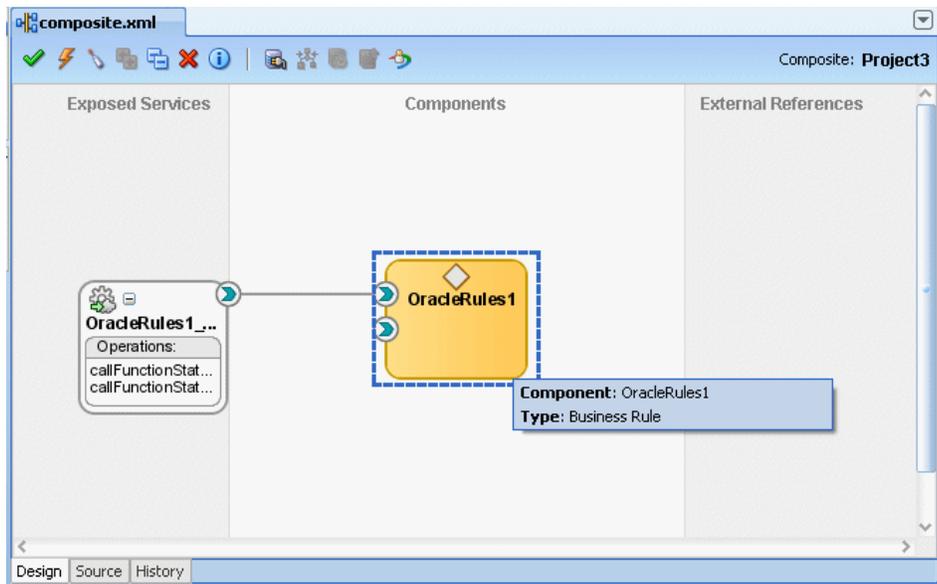
23.4.2 How to Select and Modify a Decision Function in a Business Rule Component

You can specify one or more decision functions as inputs for calling Oracle Business Rules as a component in a composite application. For example, you can specify a particular decision function as the input when multiple decision functions are available in an Oracle Business Rules dictionary.

To specify a decision function in a composite application:

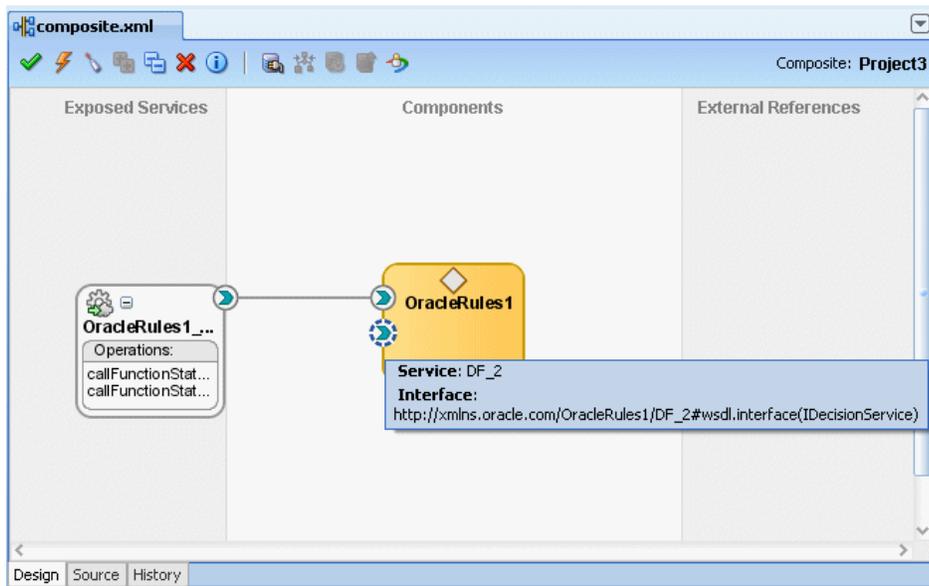
1. Add a decision function to the Oracle Business Rules dictionary. For more information, see *Oracle Fusion Middleware User's Guide for Oracle Business Rules*.
2. Add a Business Rule component to the composite application. For more information, see [Section 23.4.1, "How to Add Business Rules to an SOA Composite Application"](#).
3. Select a business rule component, as shown in [Figure 23–16](#).

Figure 23–16 *Selecting a Business Rule Component in a Composite Application*

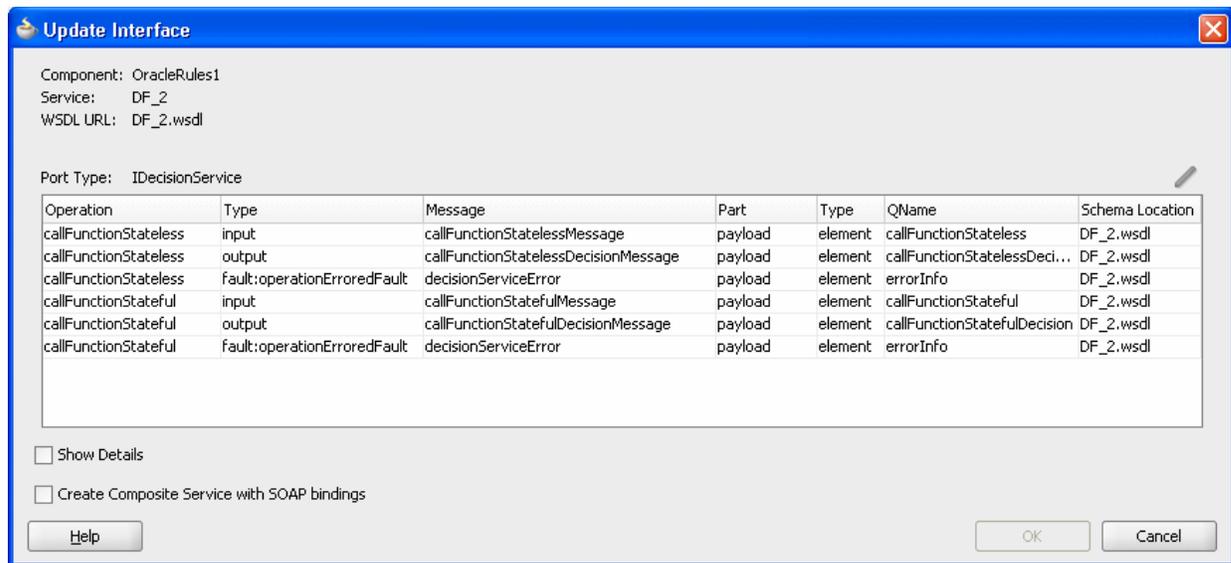


4. Select the decision function port of interest. For example, select the port for DF_2 as shown in [Figure 23–17](#).

Figure 23–17 *Selecting a Decision Function Port in a Business Rule Component*



5. When you select the port, Oracle JDeveloper shows the port information in the Property Inspector.
6. When you double-click the port, Oracle JDeveloper displays the Update Interface dialog for the port as shown in [Figure 23–18](#).

Figure 23–18 Update Interface Dialog for a Decision Function in a Business Rule Decision Port

23.5 Running Business Rules in a Composite Application

You run business rules as part of a Decision component within an SOA composite application. The business rules are executed by the Business Rule Service Engine. You can use Oracle Enterprise Manager Fusion Middleware Control Console to monitor the Business Rule Service Engine and to test an SOA composite application that includes a Decision component. For more information, see *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite*.

Part V

Using the Human Workflow Service Component

This part describes how to use the human workflow service component.

This part contains the following chapters:

- [Chapter 24, "Getting Started with Human Workflow"](#)
- [Chapter 25, "Designing Human Tasks"](#)
- [Chapter 26, "Designing Task Display Forms for Human Tasks"](#)
- [Chapter 27, "Using Oracle BPM Worklist"](#)
- [Chapter 28, "Building a Custom Worklist Client"](#)
- [Chapter 29, "Introduction to Human Workflow Services"](#)
- [Chapter 30, "Integrating Microsoft Excel with a Human Task"](#)

Getting Started with Human Workflow

This chapter introduces human workflow concepts, features, and architecture. Use cases for human workflow are provided. Instructions for designing your first workflow from start to finish are also provided.

This chapter includes the following sections:

- [Section 24.1, "Introduction to Human Workflow"](#)
- [Section 24.2, "Introduction to Human Workflow Concepts"](#)
- [Section 24.3, "Introduction to Human Workflow Features"](#)
- [Section 24.4, "Introduction to Human Workflow Architecture"](#)

For information about composite sensors, see [Chapter 51, "Defining Composite Sensors."](#)

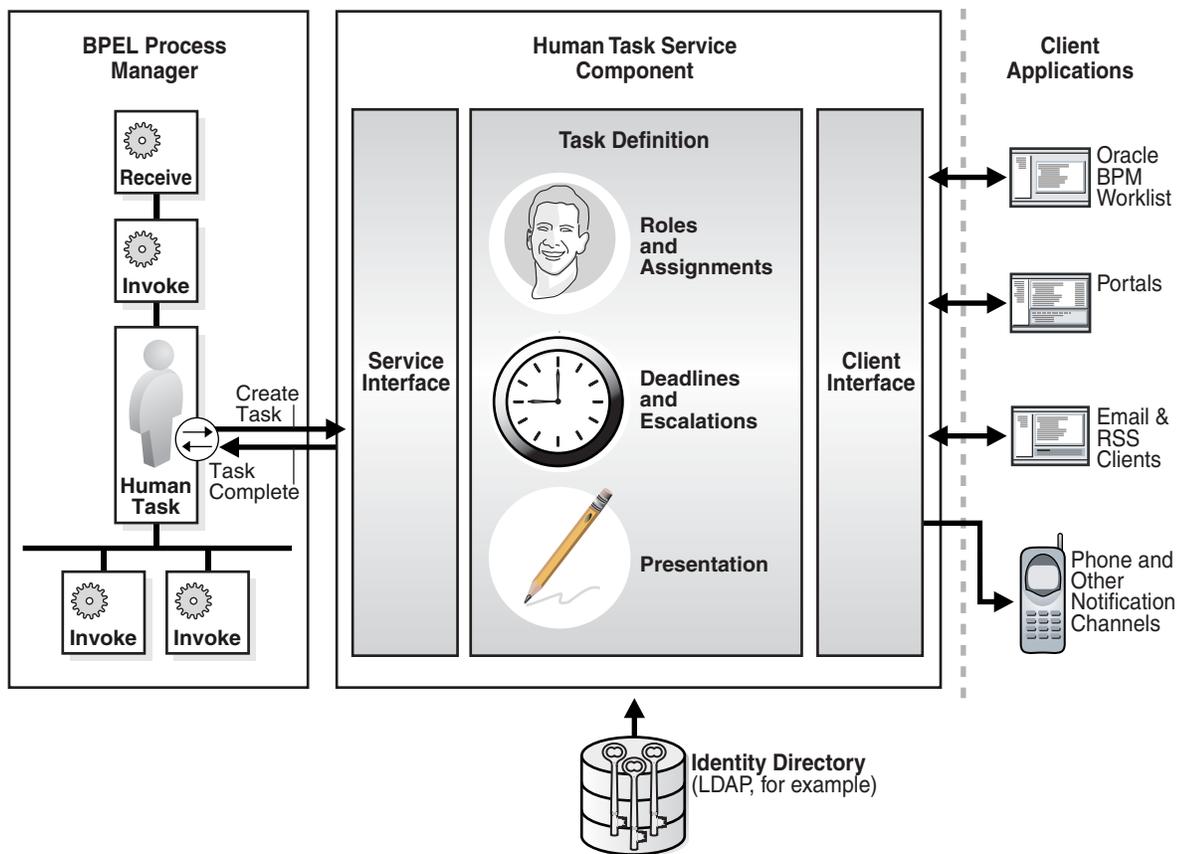
24.1 Introduction to Human Workflow

Many end-to-end business processes require human interactions with the process. For example, humans may be needed for approvals, exception management, or performing activities required to advance the business process. The human workflow component provides the following features:

- Human interactions with processes, including assignment and routing of tasks to the correct users or groups
- Deadlines, escalations, notifications, and other features required for ensuring the timely performance of a task (human activity)
- Presentation of tasks to end users through a variety of mechanisms, including a worklist application (Oracle BPM Worklist)
- Organization, filtering, prioritization, and other features required for end users to productively perform their tasks
- Reports, reassignments, load balancing, and other features required by supervisors and business owners to manage the performance of tasks

[Figure 24-1](#) provides an overview of human workflow:

Figure 24–1 Human Workflow



In Figure 24–1, the following actions occur:

- A BPEL process invokes a special activity of the human task type when it needs a human to perform a task.
- This creates a task in the human task service component. The process waits for the task to complete. It is also possible for the process to watch for other callbacks from the task and react to them.
- There is metadata associated with the task that is used by the human task service component to manage the lifecycle of the task. This includes specification of the following:
 - Who performs the task. If multiple people are required to perform the task, what is the order?
 - Who are the other stakeholders?
 - When must the task be completed?
 - How do users perform the task, what information is presented to them, what are they expected to provide, and what actions can they take?
- The human task service component uses an identity directory, such as LDAP, to determine people's roles and privileges.
- The human task service component presents tasks to users through a variety of channels, including the following:

- Oracle BPM Worklist, a role-based application that supports the concept of supervisors and process owners, and provides functionality for finding, organizing, managing, and performing tasks.
- Worklist functionality is also available as portlets that can be exposed in an enterprise portal.
- Notifications can be sent to email, phone, SMS, and other channels. Email notifications can be actionable, enabling users to perform actions on the task from within the email client without connecting to Oracle BPM Worklist or Oracle WebLogic Server.

24.2 Introduction to Human Workflow Concepts

This section introduces you to key human workflow design time and runtime concepts. This section also provides an overview of the three main stages of human workflow design.

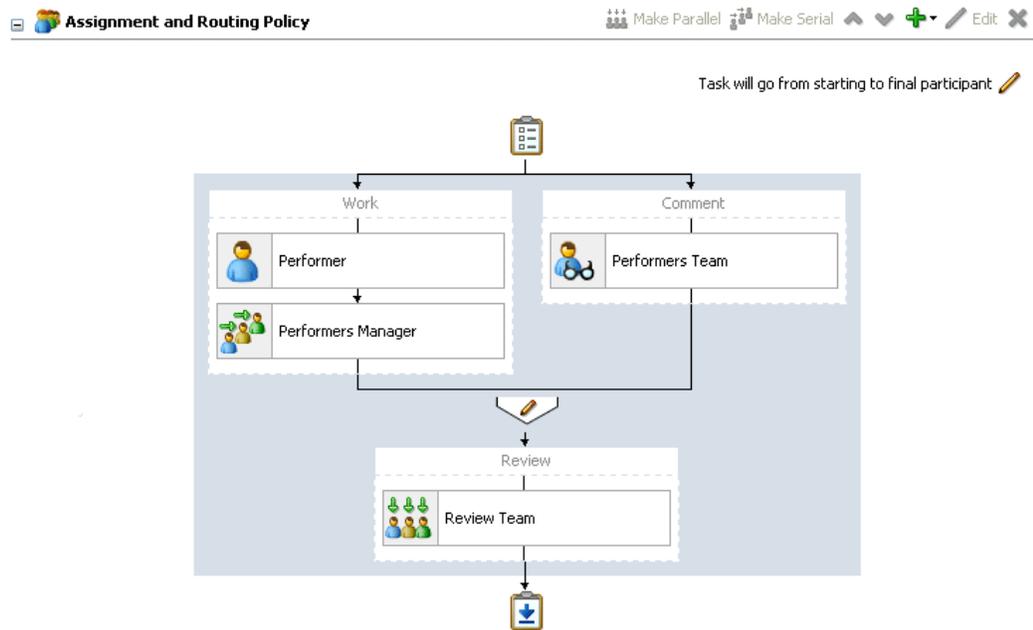
24.2.1 Introduction to Design and Runtime Concepts

Before designing a human task, it is important to understand the design and runtime concepts. A typical task consists of a subject, priority, task participants, task parameters or data, deadlines, notifications or reminders, and task forms. This section provides an overview of key concepts.

Note: Human workflow design-time tasks are performed in a graphical editor known as the Human Task Editor. The tutorial in [Section 24.3.2, "Designing a Human Task from Start to Finish"](#) describes how to use this editor.

24.2.1.1 Task Assignment and Routing

Human workflow supports declarative assignment and routing of tasks. In the simplest case, a task is assigned to a single participant (user or group). However, there are many situations in which more detailed task assignment and routing is necessary (for example, when a task must be approved by a management chain or worked and voted on by a set of people in parallel, as shown in [Figure 24–2](#)). Human workflow provides declarative pattern-based support for such scenarios.

Figure 24–2 Participants in a Task

24.2.1.1.1 Participant A participant is a user or set of users in the assignment and routing policy definition. In [Figure 24–2](#), each block with an icon representing people is a participant.

24.2.1.1.2 Participant Type In simple cases, a participant maps to a user, group, or role. However, as discussed in [Section 24.2.1.1, "Task Assignment and Routing,"](#) workflow supports declarative patterns for common routing scenarios such as management chain and group vote. The following participant types are available:

- Single approver

This is the simple case where a participant maps to a user, group, or role.

For example, a vacation request is assigned to a manager. The manager must act on the request task three days before the vacation starts. If the manager formally approves or rejects the request, the employee is notified with the decision. If the manager does not act on the task, the request is treated as rejected. Notification actions similar to the formal rejection are taken.

- Parallel

This participant indicates that a set of people must work in parallel. This pattern is commonly used for voting.

For example, multiple users in a hiring situation must vote to hire or reject an applicant. You specify the voting percentage that is needed for the outcome to take effect, such as a majority vote or a unanimous vote.

For more information, see [Section 25.3.6.2.3, "Sharing Attachments and Comments with Task Participants."](#)

- Serial

This participant indicates that a set of users must work in sequence. While working in sequence can be specified in the routing policy by using multiple participants in sequence, this pattern is useful when the set of people is dynamic.

The most common scenario for this is management chain escalation, which is done by specifying that the list is based on a management chain within the specification of this pattern.

- FYI (For Your Information)

This participant also maps to a single user, group, or role, just as in single approver. However, this pattern indicates that the participant just receives a notification task and the business process does not wait for the participant's response. FYI participants cannot directly impact the outcome of a task, but in some cases can provide comments or add attachments.

For example, a regional sales office is notified that a candidate for employment has been approved for hire by the regional manager and their candidacy is being passed onto the state wide manager for approval or rejection. FYIs cannot directly impact the outcome of a task, but in some cases can provide comments or add attachments.

For more information, see [Section 25.3.6, "How to Assign Task Participants."](#)

24.2.1.1.3 Participant Assignment A task is work that must be done by a user. When you create a task, you assign humans to participate in and act upon the task. Participants can perform actions upon tasks during runtime from Oracle BPM Worklist, such as approving a vacation request, rejecting a purchase order, providing feedback on a help desk request, or some other action. There are three types of participants:

- Users

You can assign individual users to act upon tasks. For example, you may assign users `j_london` or `j_stein` to a particular task. Users are defined in an identity store configured with the SOA Infrastructure. These users can be in the embedded LDAP of Oracle WebLogic Server, Oracle Internet Directory, or a third party LDAP directory.

- Groups

You can assign groups to act upon tasks. Groups contain individual users who can claim and act upon a task. For example, users `j_cooper` and `f_kafka` may be members of the group `LoanAgentGroup` that you assign to act upon the task.

As with users, groups are defined in the identity store of the SOA Infrastructure.

- Application roles

You can assign users who are members of application roles to claim and act upon tasks.

Application roles consist of users or other roles grouped logically for application-level authorizations. These roles are application-specific and are defined in the application Java policy store rather than the identity store. These roles are used by the application directly and are not necessarily known to a Java EE container.

Application roles define policy. Java permission can be granted to application roles. Therefore, application roles define a set of permissions granted to them directly or indirectly through other roles (if a role is granted to a role). The policy can contain grants of application roles to enterprise groups or users. In the `jazn-data.xml` file of the file-based policy store, these roles are defined in `<app-role>` elements under `<policy-store>` and written to `system-jazn-data.xml` at the farm level during deployment. You can also define these roles after deployment using Oracle Enterprise Manager Fusion

Middleware Control Console. You can set a task owner or approver to an application role at design time if the role has been previously deployed.

For more information about Oracle BPM Worklist, see [Section 24.2.1.6, "Task Forms."](#)

24.2.1.1.4 Ad Hoc Routing In processes dealing with significant variance, you cannot always determine all participants. Human workflow enables you to specify that a participant can invite other participants as part of performing the task.

For more information, see [Section 25.3.7.1.1, "Allowing All Participants to Invite Other Participants."](#)

24.2.1.1.5 Outcome-based Completion of Routing Flow By default, a task goes from starting to final participant as per the flow defined in the routing policy (as shown in [Figure 24-2](#)). However, sometimes a certain outcome at a particular step within a task's routing flow makes it unnecessary or undesirable to continue presenting the task to the next participants. For example, if an approval is rejected by the first manager, it does not need to be routed to the second manager. Human workflow supports specifying that a task or subtask be completed when a certain outcome occurs.

For more information, see [Section 25.3.7.1.2, "Stopping Routing of a Task to Further Participants."](#)

24.2.1.2 Static, Dynamic, and Rule-Based Task Assignment

There are different methods for assigning users, groups, and application roles to tasks.

- Assign tasks statically

You can assign users, groups, and application roles statically (or by browsing the identity service). The values can be either of the following:

- A single user, group, or application role (for example, `jstein`, `CentralLoanRegion`, or `ApproverRole`).
- A delimited string of users, groups, or application roles (for example, `jstein,wfaulk,cdickens`).

- Assign tasks dynamically

You can assign users, groups, and application roles dynamically using XPath expressions. These expressions enable you to dynamically determine the task participants at runtime. For example, you may have a business requirement to create a dynamic list of task approvers specified in a payload variable. The XPath expression can resolve to zero or more XML nodes. Each node value can be either of the following:

- A single user, group, or application role
- A delimited string of users, groups, or application roles. The default delimiter for the assignee delimited string is a comma (,).

For example, if the task has a payload message attribute named `po` within which the task approvers are stored, you can use the following XPath expression:

- `/task:task/task:payload/po:purchaseOrder/po:approvers`
- `ids:getManager('jstein', 'jazn.com')`

This returns the manager of `jstein`.

- `ids:getReportees('jstein', 2, 'jazn.com')`

This returns all reportees of `jstein` up to two levels.

```
ids:getUsersInGroup('LoanAgentGroup', false, 'jazn.com')
```

This returns all direct and indirect users in the group `LoanAgentGroup`.

- Assign tasks with business rules

You can create the list of task participants with complex expressions. The result of using business rules is equal to using XPath expressions.

24.2.1.3 Task Stakeholders

A task has multiple stakeholders. Participants are the users defined in the assignment and routing section of the task definition. These users are the primary stakeholders that perform actions on the task.

In addition to the participants specified in the assignment and routing policy, human workflow supports additional stakeholders:

- Owner

This participant has business administration privileges on the task. This participant can be specified as part of the task definition or from the invoking process (and for a particular instance). The task owner can act upon tasks they own and also on behalf of any other participant. The task owner can change both the outcome of the task and the assignments.

For more information, see [Section 25.3.4.6, "Specifying a Task Owner"](#) to specify an owner in the Human Task Editor or [Section 25.4.4.2, "Specifying a Task Owner"](#) to specify an owner in the **Advanced** tab of the Create Human Task dialog.

- Initiator

The person who initiates the process (for example, the initiator files an expense report for approval). This person can review the status of the task using initiated task filters. Also, a useful concept is for including the initiator as a potential candidate for request-for-information from other participants.

For more information, see [Section 25.4.3.2, "Specifying the Task Initiator and Task Priority."](#)

- Reviewer

This participant can review the status of the task and add comments and attachments.

- Admin

This participant can view all tasks and take certain actions such as reassigning a test, suspending a task to handle errors, and so on. The task admin cannot change the outcome of a task.

While the task admin cannot perform the types of actions that a task participant can, such as approve, reject, and so on, this participant type is the most powerful because it can perform actions such as reassign, withdraw, and so on.

- Error Assignee

When an error occurs, the task is assigned to this participant (for example, the task is assigned to a nonexistent user). The error assignee can perform task recovery actions from Oracle BPM Worklist, the task display form in which you perform task actions during runtime.

For more information, see [Section 25.3.7.4, "Configuring the Error Assignee."](#)

24.2.1.4 Task Deadlines

Human workflow supports the specification of deadlines associated with a task. You can associate the following actions with deadlines:

- Reminders:
The task can be reminded multiple times based on the time after the assignment or the time before the expiration.
- Escalation:
The task is escalated up the management hierarchy.
- Expiration:
The task has expired.
- Renewal:
The task is automatically renewed.

For more information, see [Section 25.3.8, "How to Escalate, Renew, or End the Task."](#)

24.2.1.5 Notifications

You can configure your human task to use notifications. Notifications enable you to alert interested users to changes in the state of a task during the task life cycle. For example, a notification is sent to an assignee when a task has been approved or withdrawn.

You can specify for notifications to be sent to different types of participants for different actions. For example, you can specify the following:

- For the owner of a task to receive a notification message when a task is in error (for example, been sent to a nonexistent user).
- For a task assignee to receive a notification message when a task has been escalated.

You can specify the contents of the notification message and the notification channel to use for sending the message.

- Email
You can configure email notification messages to be actionable, meaning that a task assignee can act upon a task from within the email.
- Voice message
- Instant messaging (IM)
- Short message service (SMS)

For example, you may send the following message by email when a task assignee requests additional information before they can act upon a task:

```
In order for me to approve this task, I need more information to justify the need  
for this business trip
```

During runtime, you can mark a message sender's address as spam and also display a list of bad or invalid addresses. These addresses are automatically removed from the bad address list.

For more information about notifications, see the following:

- [Section 25.3.9, "How to Specify Participant Notification Preferences"](#)

- [Chapter 16, "Using the Notification Service"](#)
- [Part VII, "Using Oracle User Messaging Service"](#)

24.2.1.6 Task Forms

Task display forms provide you with a way to interact with a task. Oracle BPM Worklist displays all worklist tasks that are assigned to task assignees in the task display form. When you drill down into a specific task, the task display form displays the contents of the task to the user's worklist. For example, an expense approval task may show a form with line items for various expenses, and a help desk task form may show details such as severity, problem location, and so on.

The integrated development environment of Oracle SOA Suite includes Oracle Application Development Framework (Oracle ADF) for this purpose. With Oracle ADF, you can design a task display form that depicts the human task in the SOA composite.

ADF-based task display forms can be automatically generated. Advanced users can design their own task display forms by using ADF data controls to lay out the content on the page and connect to the workflow service engine at execution time to retrieve task content and act on tasks.

You can create task forms in JSF, .NET, or any other client technologies using the APIs. Integration with Microsoft Excel for initiating and acting on tasks is also provided.

For more information, see the following:

- [Chapter 26, "Designing Task Display Forms for Human Tasks."](#)
- [Chapter 27, "Using Oracle BPM Worklist"](#)

24.2.1.7 Advanced Concepts

This section describes advanced human workflow concepts.

24.2.1.7.1 Rule-based Routing You can use Oracle Business Rules to dynamically alter the routing flow. If used, each time a participant completes their step, the associated rules are invoked and the routing flow can be overridden from the rules.

For more information, see [Section 25.3.7.2, "Specifying Advanced Task Routing Using Business Rules."](#)

24.2.1.7.2 Rule-based Participant Assignment You can use Oracle Business Rules to dynamically build a list of users, groups, and roles to be associated with a participant.

For more information, see [Section 25.3.6, "How to Assign Task Participants."](#)

24.2.1.7.3 Stages A stage is a way of organizing the approval process for blocks of participant types. You can have one or more stages in sequence or in parallel. Within each stage, you can have one or more participant type blocks in sequence or in parallel.

For more information, see [Section 25.3.6, "How to Assign Task Participants."](#)

24.2.1.7.4 Access Rules You can specify access rules that determine the parts of a task that assignees can view and update. For example, you can configure the task payload data to be read by assignees. This action enables only assignees (and nobody else) to have read permissions. No one, including assignees, has write permissions.

For more information, see [Section 25.3.10.8, "Specifying Access Policies on Task Content."](#)

24.2.1.7.5 Callbacks While human workflow supports detailed behavior that can be declaratively specified, in some advanced situations, more extensible behavior may be required. Task callbacks enable such extensibility; these callbacks can either be handled in the invoking BPEL process or a Java class.

For more information, see [Section 25.3.10.5, "Specifying Callback Classes on Task Status."](#)

24.2.1.8 Reports and Audit Trails

Oracle BPM Worklist provides several out-of-the-box reports for task analysis:

- Unattended tasks
Analysis of tasks assigned to users' groups or reportees' groups that have not yet been acquired.
- Tasks priority
Analysis of tasks assigned to a user, reportees, or their groups, based on priority.
- Tasks cycle time
Analysis of the time taken to complete tasks from assignment to completion based on users' groups or reportees' groups.
- Tasks productivity
Analysis of assigned tasks and completed tasks in a given time period for a user, reportees, or their groups.
- Tasks time distribution
The time an assignee takes to perform a task.

You can view an audit trail of actions performed by the participants in the task and a snapshot of the task payload and attachments at various points in the workflow. The short history for a task lists all versions created by the following tasks:

- Initiate task
- Reinitiate task
- Update outcome of task
- Completion of task
- Erring of task
- Expiration of task
- Withdrawal of task
- Alerting of task to the error assignee

For more information, see [Chapter 27, "Using Oracle BPM Worklist."](#)

24.2.2 Introduction to the Stages of Human Workflow Design

Human workflow modeling consists of three stages of modeling, as described in [Table 24-1](#).

Table 24–1 Stages of Human Workflow Modeling

Step	Description	For More Information...
1	You create and define contents of the human task in the Human Task Editor, including defining a participant type, routing policy, escalation and expiration policy, notification, and so on.	Section 25.2.1, "Create a Human Task Definition."
2	You associate the human task definition with a BPEL process. The BPEL process integrates a series of activities (including the human task activity) and services into an end-to-end process flow.	Section 25.2.2, "Associate the Human Task Definition with a BPEL Process."
3	You create a task display form. This form is used for displaying the task details on which you act at runtime in Oracle BPM Worklist.	Section 25.2.3, "Generate the Task Display Form."

24.3 Introduction to Human Workflow Features

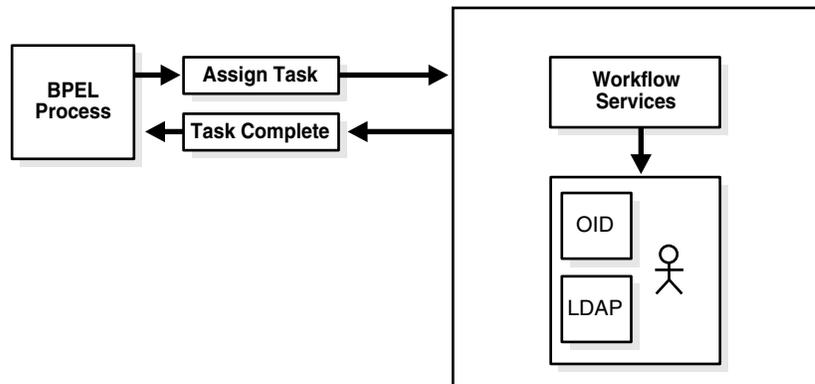
This section provides an introduction to use cases for human workflow. After that, a tutorial is provided that guides you through the design of a human task from start to finish.

24.3.1 Human Workflow Use Cases

The following sections describe multiple use cases for workflow services.

24.3.1.1 Task Assignment to a User or Role

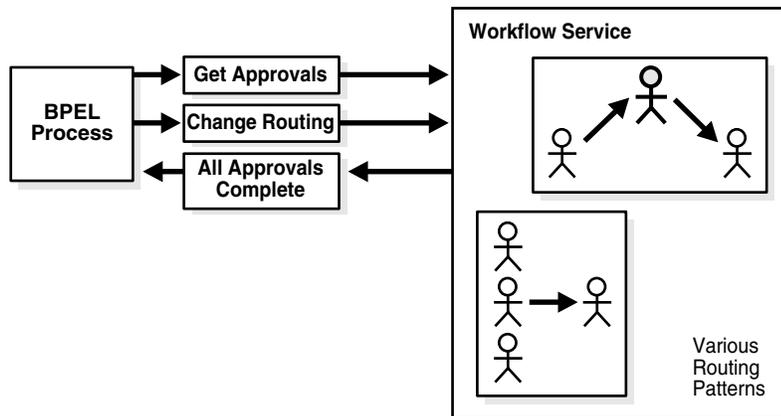
A vacation request process may start with getting the vacation details from a user and then routing the request to their manager for approval. User details and the organizational hierarchy can be looked up from a user directory or identity store. This scenario is shown in [Figure 24–3](#).

Figure 24–3 Assigning Tasks to a User or Role from a Directory

24.3.1.2 Use of the Various Participant Types

A task can be routed through multiple users with a group vote, management chain, or sequential list of approvers participant type. For example, consider a loan request that is part of the loan approval flow. The loan request may first be assigned to a loan agent role. After a specific loan agent acquires and accepts the loan, the loan may be routed further through multiple levels of management if the loan amount is greater than \$100,000. This scenario is shown in [Figure 24–4](#).

Figure 24–4 Flow Patterns and Routing Policies

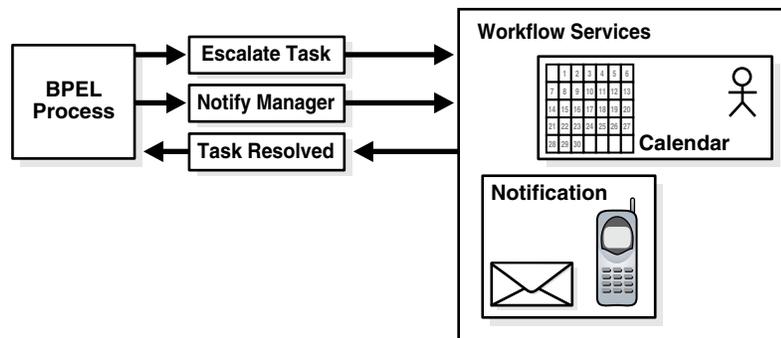


You can use these types as building blocks to create complex workflows.

24.3.1.3 Escalation, Expiration, and Delegation

A high-priority task can be assigned to a certain user or role based on the task type through use of custom escalation functions. However, if the user does not act on it in a certain time, the task may expire and in turn be escalated to the manager for further action. As part of the escalation, you may also notify the users by email, telephone voice message, or SMS. Similarly, a manager may delegate tasks from one reportee to another to balance the load between various task assignees. All tasks defined in BPEL have an associated expiration date. Additionally, you may specify escalation or renewal policies, as shown in Figure 24–5. For example, consider a support call, which is part of a help desk service request process. A high-priority task may be assigned to a certain user and if the user does not respond in two days, the task is routed to the manager for further action.

Figure 24–5 Escalation and Notification



24.3.1.4 Automatic Assignment and Delegation

A user may decide to have another user perform tasks on their behalf. Tasks can be explicitly delegated from the Oracle BPM Worklist or can be automatically delegated. For example, a manager sets up a vacation rule saying that all their high priority tasks are automatically routed to one of their direct reports while the manager is on vacation. In some cases, tasks can be routed to different individuals based on the content of the task. Another example of automatic routing is to allocate tasks among multiple individuals belonging to a group. For example, a help desk supervisor decides to allocate all tasks for the western region based on a round robin basis or

assign tasks to the individual with the lowest number of outstanding tasks (the least busy).

24.3.1.5 Dynamic Assignment of Users Based on Task Content

An employee named James in the human resources department requests new hardware that costs \$5000. The company may have a policy that all hardware expenses greater than \$3000 must go through manager and vice president approval, and then review by the director of IT. In this scenario, the workflow can be configured to automatically determine the manager of James, the vice president of the human resources department, and the director of IT. The purchase order is routed through these three individuals for approval before the hardware is purchased.

24.3.2 Designing a Human Task from Start to Finish

This section guides you through design of your first human task.

This sample describes how an employee submits a vacation request that is automatically routed to their manager for approval. Once the manager responds (approved or rejected), a notification is sent to the employee.

This sample illustrates creation of a SOA composite application with two components:

- A BPEL process
- A human task, for approving a vacation request submitted by an employee

This example highlights the use of the following:

- Using the SOA Composite Editor
- Modeling a single approval workflow using Oracle BPEL Designer
- Creating an Oracle ADF-based Oracle BPM Worklist
- Using Oracle BPM Worklist to view and respond to the task

24.3.2.1 Prerequisites

This tutorial makes the following assumptions:

- Oracle SOA Suite is installed on a host on which the SOA Infrastructure is configured.
 - You are familiar with basic BPEL constructs, including BPEL activities and partner links, and basic XPath functions. Familiarity with the SOA Composite Editor and Oracle BPEL Designer, the environment for designing and deploying BPEL processes, is also assumed.
1. Create a file named `VacationRequest.xsd` with the following syntax. This file includes the schema for the vacation request and subsequent response.

```
<schema attributeFormDefault="qualified" elementFormDefault="qualified"
  targetNamespace="http://xmlns.oracle.com/VacationRequest"
  xmlns="http://www.w3.org/2001/XMLSchema">
  <element name="VacationRequestProcessRequest">
    <complexType>
      <sequence>
        <element name="creator" type="string"/>
        <element name="fromDate" type="date"/>
        <element name="toDate" type="date"/>
        <element name="reason" type="string"/>
      </sequence>
    </complexType>
  </element>
</schema>
```

```
</element>
<element name="VacationRequestProcessResponse">
  <complexType>
    <sequence>
      <element name="result" type="string"/>
    </sequence>
  </complexType>
</element>
</schema>
```

Note: The `VacationRequest.xsd` file is also available for download as part of tutorial `workflow-100-VacationRequest`. See [Section 24.3.3, "Additional Tutorials"](#) for information on downloading this and other tutorials.

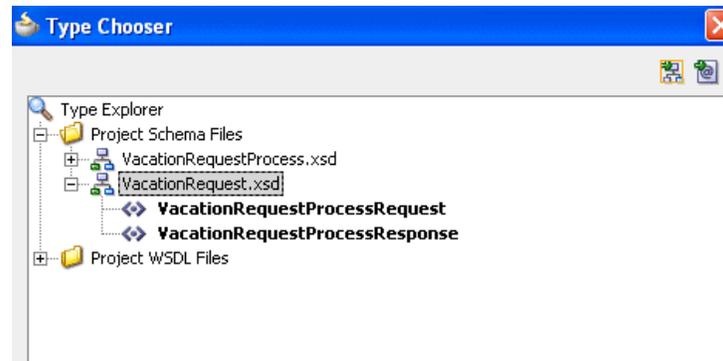
24.3.2.2 How to Create the Vacation Request Process

In this tutorial, you create a new application and SOA project and design the human task to send a vacation request to a manager for approval or rejection. You also create a second application and project in which you create an Oracle ADF-based task display form from which to act upon the vacation request.

24.3.2.2.1 Creating an Application and a Project with a BPEL Process

To create an application and a project with a BPEL process:

1. Start Oracle JDeveloper.
2. From the **File** main menu, select **New > Applications > SOA Application**.
3. Click **OK**.
4. In the **Application Name** field, enter `VacationRequest`, and click **Next**.
5. In the **Project Name** field, enter `VacationRequest`, and click **Next**.
6. In the **Composite Template** list, select **Composite with BPEL**, and click **Finish**.
7. The Create BPEL Process dialog appears.
8. In the **Name** field, enter `VacationRequestProcess`.
9. Go to the bottom of the Create BPEL Process dialog.
10. To the right of the **Input** field, click the **Search** icon.
The Type Chooser dialog appears.
11. In the upper right corner, click the **Import Schema File** icon.
The Import Schema dialog appears.
12. Browse for and select the `VacationRequest.xsd` file you created in [Section 24.3.2.1, "Prerequisites."](#)
13. Click **OK** until you are returned to the Type Chooser dialog, as shown in [Figure 24-6](#).

Figure 24–6 Type Chooser Dialog with the Request and Response Elements

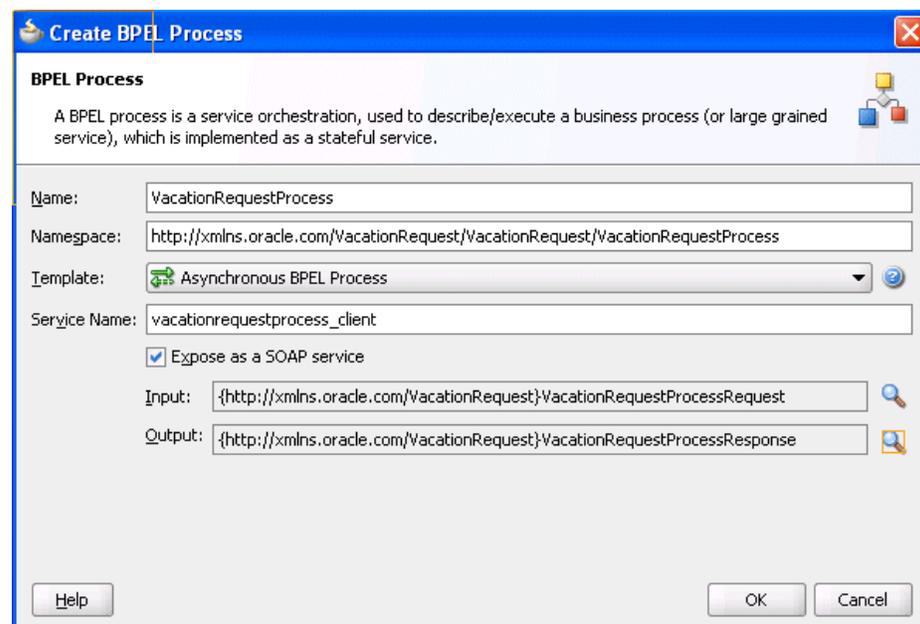
14. Select the input element **VacationRequestProcessRequest**, and click **OK**.

You are returned to the Create BPEL Process dialog.

15. To the right of the **Output** field, click the **Search** icon.

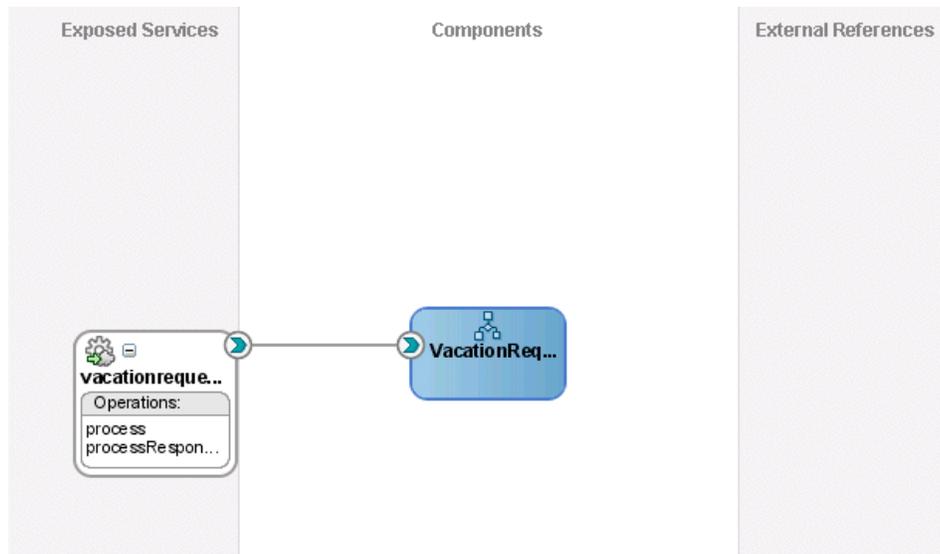
16. Select the output element **VacationRequestProcessResponse**, and click **OK**.

You are returned to the Create BPEL Process dialog, as shown in [Figure 24–7](#).

Figure 24–7 BPEL Process Dialog

17. Accept the default values for all other settings, and click **OK**.

A BPEL process service component is created in the SOA Composite Editor. Because **Expose as a SOAP service** was selected in the Create BPEL Process dialog, the BPEL process is automatically connected with a service binding component. The service exposes the SOA composite application to external customers.

Figure 24–8 BPEL Process in SOA Composite Editor

For more information about service components and the SOA Composite Editor, see [Chapter 4, "Introduction to the Functionality of the SOA Composite Editor."](#)

24.3.2.2.2 Create the Human Task Service Component

You are now ready to create the human task service component in which you design your human task.

To create the human task service component:

1. From the SOA list of the Component Palette, drag a **Human Task** into the SOA Composite Editor.

The Create Human Task dialog appears.

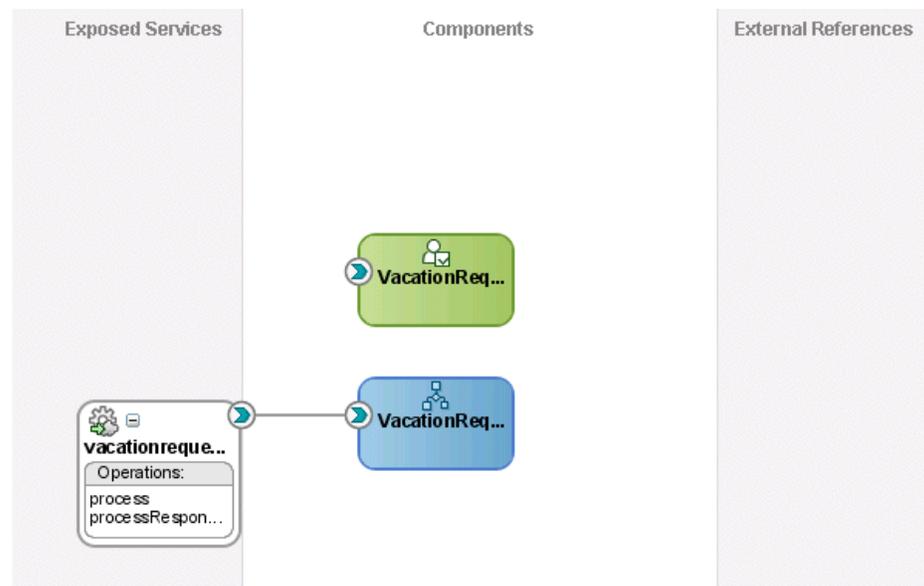
2. Enter the details described in [Table 24–2](#).

Table 24–2 Create Human Task Dialog Fields and Values

Field	Value
Name	Enter VacationRequestTask.
Namespace	Accept the default value.
Create Composite Service with SOAP Bindings	Do <i>not</i> select the checkbox. Instead, you create a human task that you later associate with the BPEL process you created in Section 24.3.2.2.1, "Creating an Application and a Project with a BPEL Process." The BPEL process was created with an automatically-bound web service.

3. Click **OK**.

The **Human Task** icon appears in the SOA Composite Editor above the BPEL process, as shown in [Figure 24–9](#).

Figure 24–9 Human Task Icon in SOA Composite Editor

4. Double-click the **Human Task** icon.

The Human Task Editor appears. You are now ready to begin design of your human task.

24.3.2.2.3 Designing the Human Task

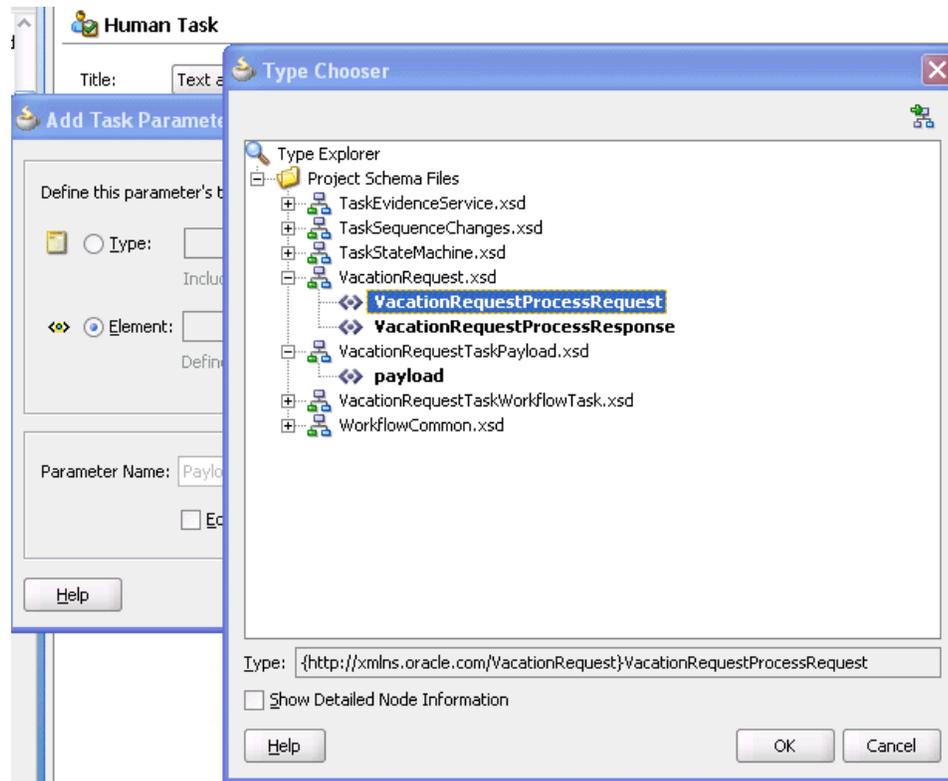
To design the human task:

1. In the **Title** field, enter `Request for Vacation`.
2. Accept the default values for outcomes (**APPROVE** and **REJECT**). For this task, these outcomes represent the two choices the manager has for acting on the vacation request.
3. On the right side of the **Parameters** section, click the **Add** icon to specify the task payload.

The Add Task Parameter dialog is displayed. You now create parameters to represent the elements in your XSD file. This makes the payload data available to the workflow task.

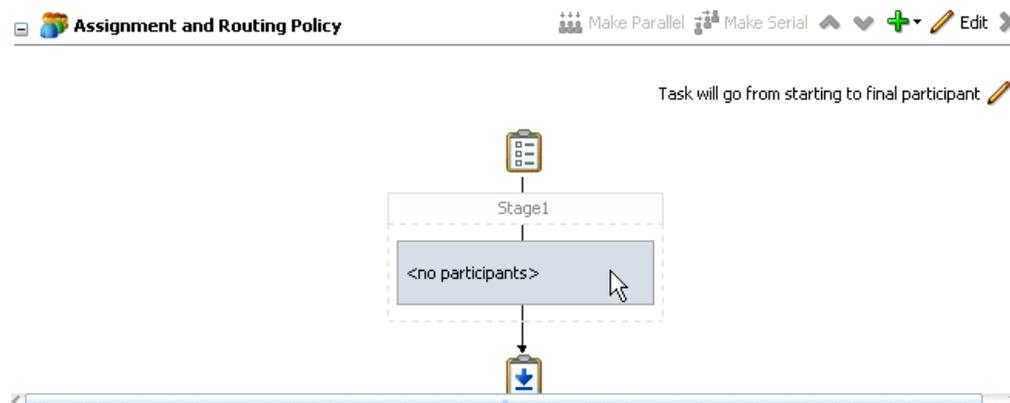
4. Select **Element**.
5. To the right of the **Element** field, click the **Search** icon.
The Type Chooser dialog appears.
6. Expand and select **Project Schema Files > VacationRequest.xsd > VacationRequestProcessRequest**, and click **OK**. [Figure 24–10](#) provides details.

Figure 24–10 Type Chooser Dialog



7. Ensure that the **Editable via worklist** checkbox is selected. This provides you with the option to modify this parameter during runtime from Oracle BPM Worklist.
8. Click **OK** on the Add Task Parameter dialog.
9. In the **Assignment and Routing Policy** section, highlight the **<no participants>** box below **Stage1**, as shown in Figure 24–11.

Figure 24–11 Assignment and Routing Policy



10. On the right side of the **Assignment and Routing Policy** section, click the **Edit** icon.

The Edit Participant Type dialog appears. You now add participants to this task. As described in Section 24.2.1.1.2, "Participant Type," Oracle SOA Suite provides

several out-of-the-box patterns known as participant types for addressing specific business needs.

11. Accept the default participant type of **Single** that displays in the **Type** list. You select this type because a single assignee, the manager, acts on the vacation request task.

12. In the **Participant Name** table, click the **Add** icon, and select **Add User**.

This participant type acts alone on the task.

13. Click the **Data Type** column, and select **By Expression** from the list that is displayed. [Figure 24–12](#) provides details.

This action enables the task to be assigned dynamically by the contents of the task. The employee filing the vacation request comes from the parameter passed to the task (the `creator` element in the XSD file you imported in [Section 24.3.2.2.1, "Creating an Application and a Project with a BPEL Process"](#)). The task is automatically routed to the employee's manager.

Figure 24–12 Selection of By Expression from the Data Type Column

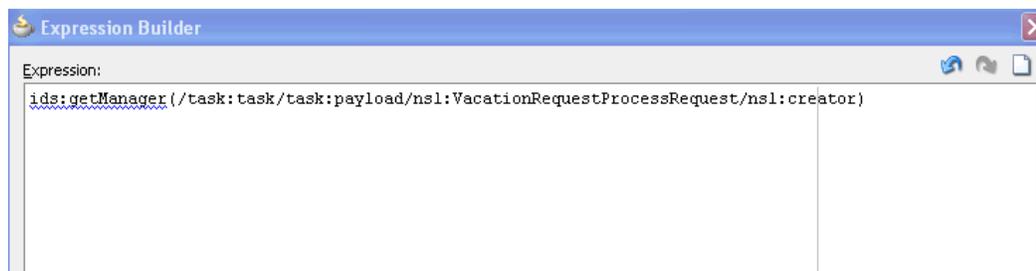


14. In the **Value** column, click the **Browse** icon (the dots) to invoke the Expression Builder dialog.
15. In the dropdown list in the **Functions** section, select **Identity Service Functions**.
16. Select `getManager`. This function gets the manager of the user who created the vacation request task.
17. Above the **Functions** section, click **Insert into Expression**.
18. Place the cursor between the parentheses of the function.
19. In the **Schema** section, expand `task:task > task:payload > ns1:VacationRequestProcessRequest > ns1:creator`.

where `ns1` is the namespace for this example; your namespace may be different.

20. Click **Insert into Expression**.

The Expression Builder dialog displays the XPath expression in the **Expression** section. [Figure 24–13](#) provides details.

Figure 24–13 XPath Expression

21. Click **OK** to exit the Expression Builder dialog.
22. From the **File** menu, select **Save All**.
23. Click **OK** to exit the Add Participant Type dialog.

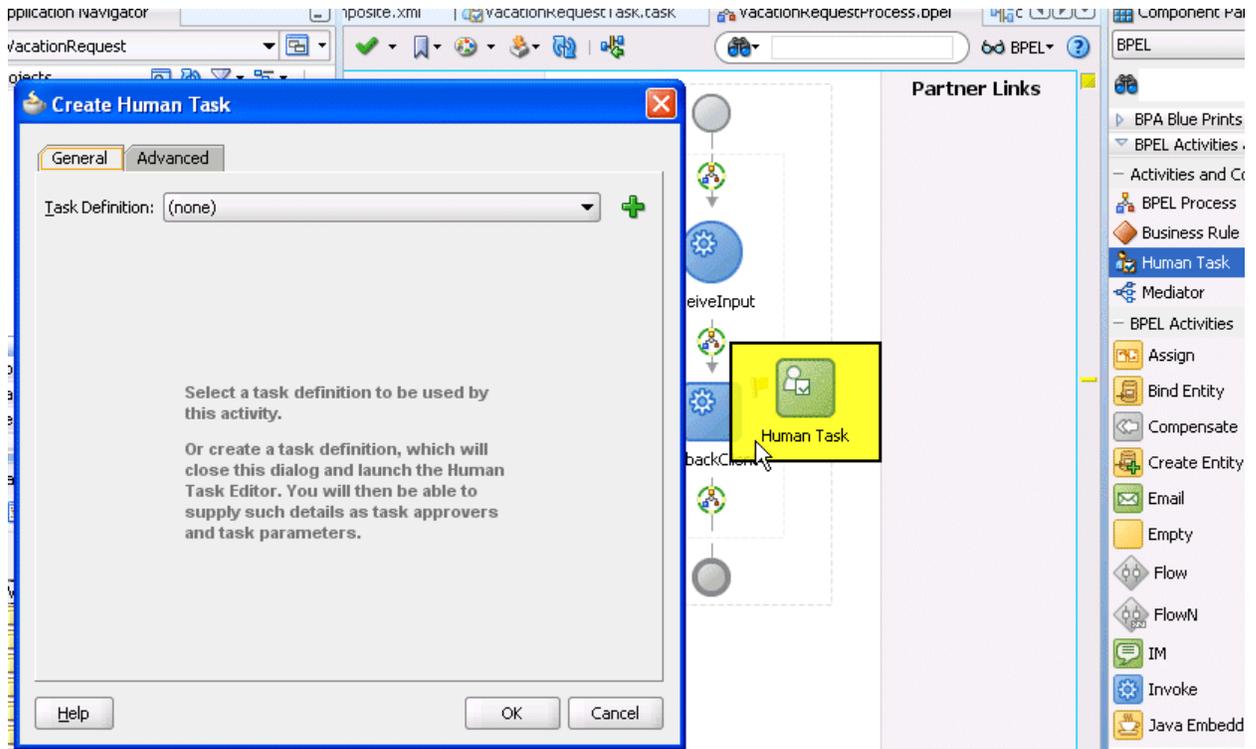
24.3.2.2.4 Associating the Human Task and BPEL Process Service Components

You are now ready to associate your human task with the BPEL process you created in [Section 24.3.2.2.1, "Creating an Application and a Project with a BPEL Process."](#)

To associate the human task and BPEL process service component:

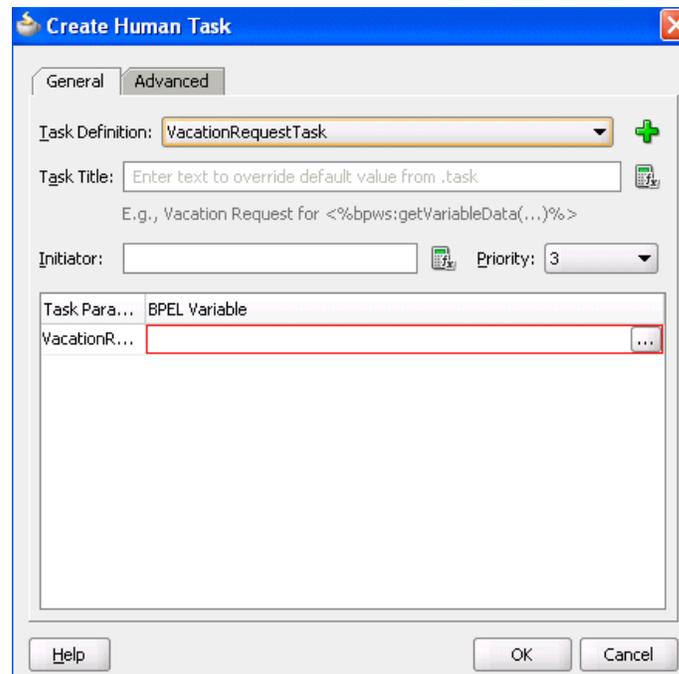
1. In the Application Navigator, double-click **composite.xml**.
2. Double-click the **VacationRequestProcess** BPEL process service component in the SOA Composite Editor.
The BPEL process displays in Oracle BPEL Designer.
3. From the list at the top of the Component Palette, select **BPEL**.
4. Expand **BPEL Activities and Components**.
5. Drag a **Human Task** beneath the **receiveInput** receive activity.

The Create Human Task dialog appears, as shown in [Figure 24–14](#).

Figure 24–14 Human Task Creation

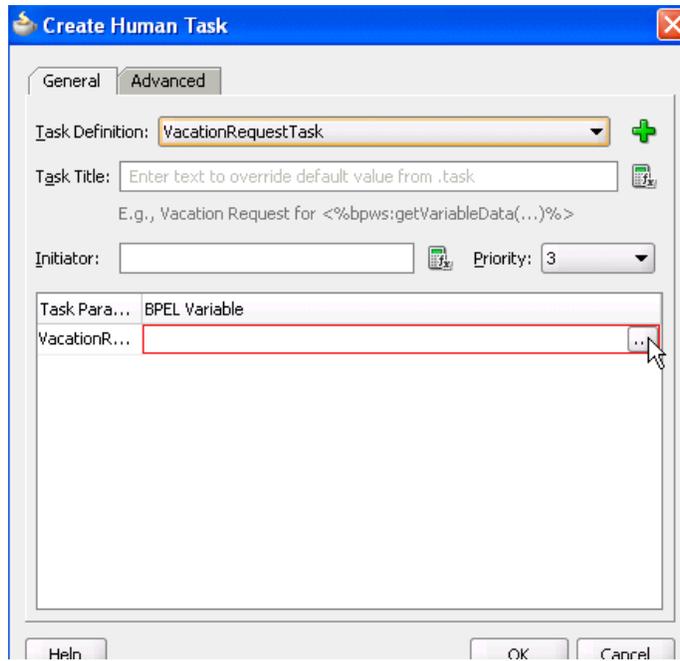
- From the **Task Definition** list, select the **VacationRequestTask** task you created (if it is not currently displaying).

The dialog refreshes as shown in [Figure 24–15](#) to display additional fields.

Figure 24–15 Create Human Task Dialog

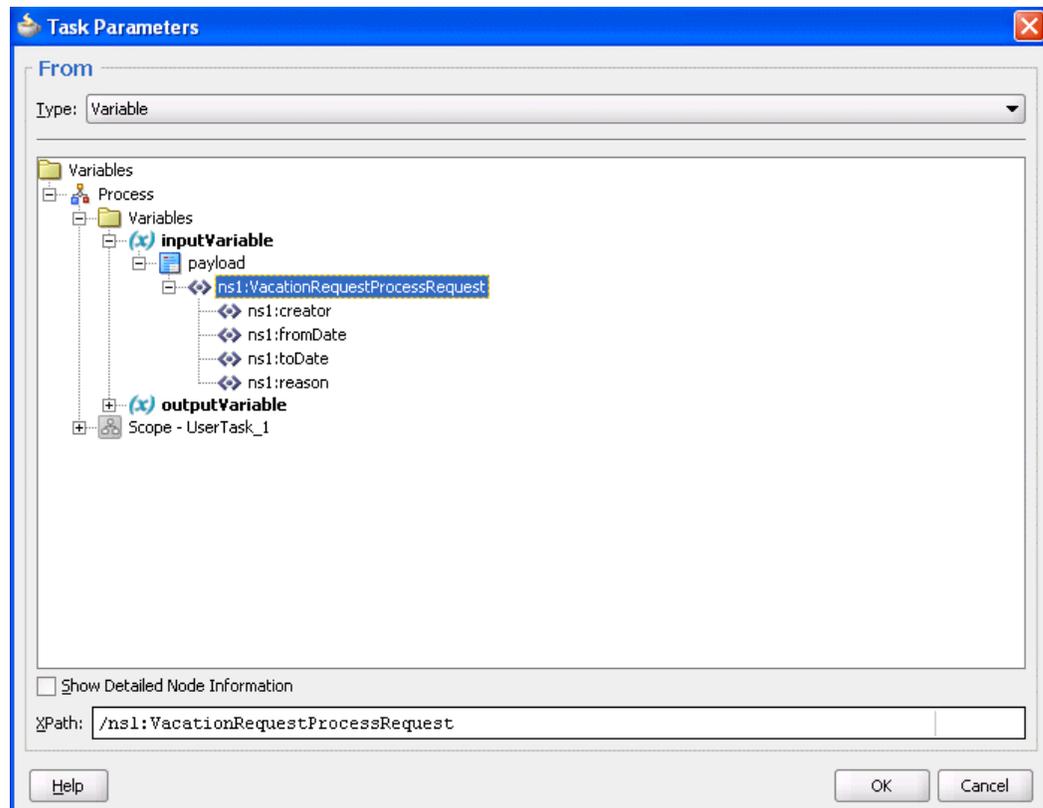
7. In the **BPEL Variable** column, click the **Browse** icon (dots) shown in [Figure 24–16](#) for the **requester** parameter.

Figure 24–16 BPEL Variable Entry



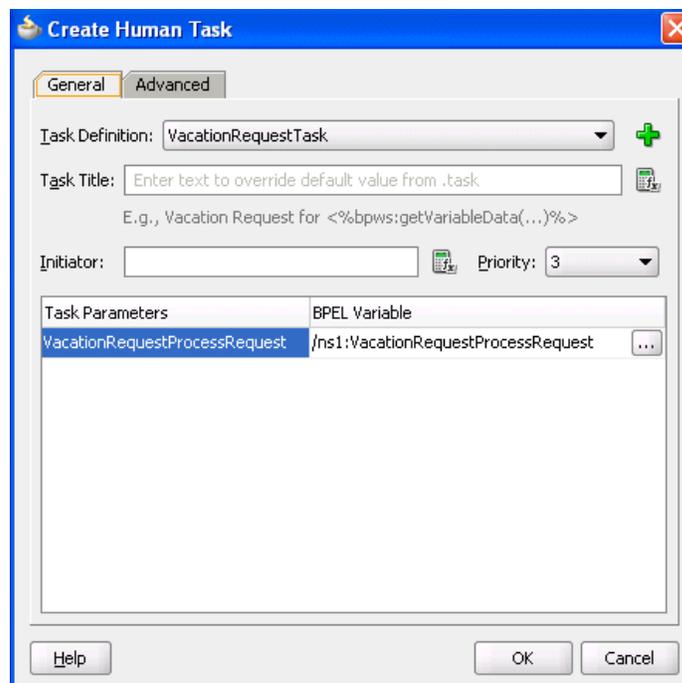
The Task Parameters dialog appears.

8. From the **Type** list, select **Variable**.
9. Expand **Process > Variables > inputVariable > payload > ns1:VacationRequestProcessRequest**. [Figure 24–17](#) provides details.

Figure 24–17 Variable Selection

10. Click OK.

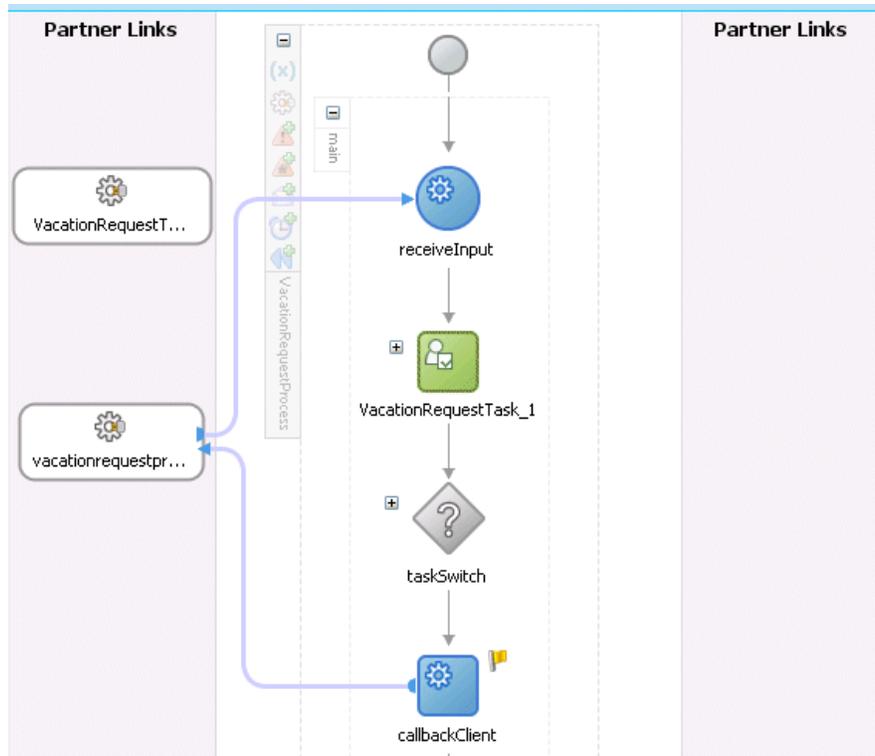
When complete, the dialog looks as shown in [Figure 24–18](#):

Figure 24–18 BPEL Variable

11. Click **OK** to close the Create Human Task dialog.

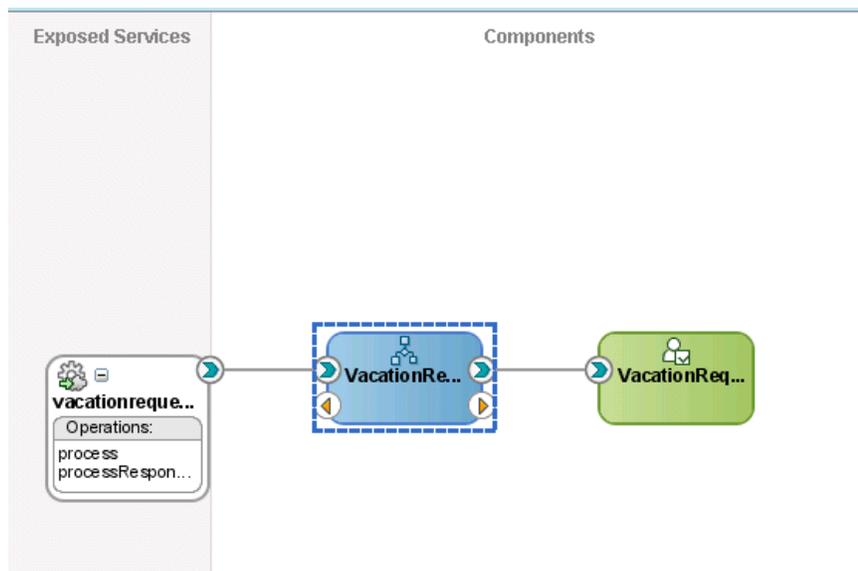
The human task activity and request and response partner links now appear.

Figure 24–19 Human Task and Partner Links in Oracle BPEL Designer



12. Return to the SOA Composite Editor and note that the BPEL process and human task service components have been automatically connected.

Figure 24–20 SOA Composite Editor



13. From the **File** menu, select **Save All**.

24.3.2.2.5 Creating an Application Server Connection

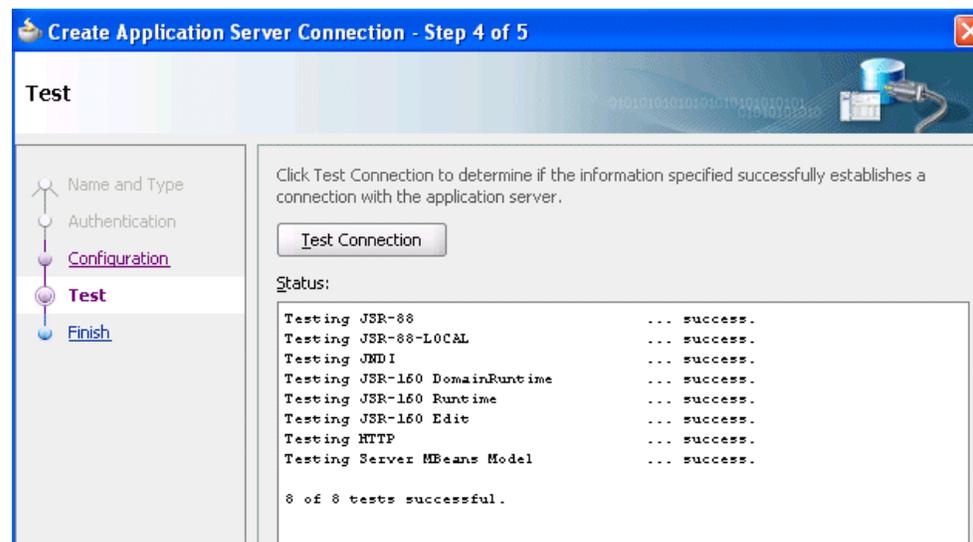
You are now ready to create a connection to the application server on which Oracle SOA Suite is installed and configured with the SOA Infrastructure.

To create an application server connection

1. From the **File** main menu, select **New > Connections > Application Server Connection**.
2. Click **OK**.
3. In the **Connection Name** field, enter a connection name.
4. From the **Connection Type** list, select **WebLogic 10.3**.
5. Click **Next**.
6. In the **Username** field, enter `weblogic`.
7. In the **Password** field, enter the password for connecting to the application server.
8. Click **Next**.
9. Enter the hostname for the application server that is configured with the SOA Infrastructure.
10. In the **WLS Domain** field, enter the Oracle WebLogic Server domain.
11. Click **Next**.
12. Click **Test Connection**.

If successful, the message shown in [Figure 24–21](#) is displayed.

Figure 24–21 Connection Success



13. Click **Finish**.
14. From the **File** menu, select **Save All**.

24.3.2.2.6 Deploying the SOA Composite Application

You are now ready to deploy to the application server on which you created the connection.

To deploy the SOA composite application

1. In the Application Navigator, right-click the **VacationRequest** project and select **Deploy > VacationRequest > application_server_connection_name**.

The SOA Deployment Configuration dialog appears.

2. Select the target server, and click **OK**.

The project is deployed.

24.3.2.2.7 Initiating the Process Instance

To initiate the process instance:

1. See *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite* for instructions on accessing the Test Web Service page for initiating the process instance.

24.3.2.2.8 Creating a Task Display Form Project

You are now ready to create a project for the task display form. This is a separate project from the one in which you created the human task.

To create a task display form project:

1. Double-click the **VacationRequestProcess** BPEL process.
2. Right click the **VacationRequestTask_1** human task activity in Oracle BPEL Designer.

3. Select **Auto-Generate Task Form**.

The Create Project dialog appears.

4. In the **Project Name** field, enter `VacationRequestTaskFlow`, and click **OK**.
5. From the **File** main menu, select **Save All**.

24.3.2.2.9 Acting on the Task in Oracle BPM Worklist

To resolve the task in Oracle BPM Worklist:

1. Go to Oracle BPM Worklist:

`http://hostname:7001/integration/worklistapp`

2. Log in to Oracle BPM Worklist.
3. Resolve the task.

24.3.2.2.10 Deploying the Task Display Form

To deploy the task display form:

1. In the Application Navigator, right-click the **VacationRequestTaskFlow** project and select **Deploy > to > VacationRequestTaskFlow > application_server_connection_name**.

The SOA Deployment Configuration dialog appears.

2. Select the target server, and click **OK**.

The task form is deployed.

3. Return to Oracle BPM Worklist.

4. Note that the task display form now appears at the bottom of Oracle BPM Worklist.

24.3.3 Additional Tutorials

In addition to the vacation request use case, other tutorials are available at the following URL:

http://www.oracle.com/technology/sample_code/products/hwf

Table 24–3 provides an overview of some samples. All samples show the use of worklist applications and workflow notifications. For the complete list of samples, visit the URL.

Table 24–3 End-to-End Examples

Sample	Description	Name
Vacation Request	Provides a sample in which a user submits a vacation request that gets assigned to their manager for approval or rejection. This sample also describes how to create Oracle ADF task forms for the vacation request to act on the task.	workflow-100-VacationRequest
Help Desk Request	Provides a simple workflow sample using Oracle ADF task forms for task approval.	workflow-101-HelpDeskRequest
Sales Quote Request	Provides a complex workflow sample with chaining of multiple tasks.	workflow-102-SalesQuote
Expense Application	Provides a sample that integrates workflow with Oracle ADF Business Components. Events are raised to the BPEL process and the human workflow is invoked for task approval.	workflow-103-ExpenseApp
Contract Approval	Provides a sample of approving a contract. This sample uses digital signatures for tasks.	workflow-104-ContractApproval
Document Workflow	Provides a sample in which a document is reviewed by a group of participants in parallel. In the end, voting determines if the document is approved or rejected.	workflow-105-documentworkflow
Iterative Design	Provides a sample in which a workflow task can be passed multiple times between assignees during the design process. Advanced routing rules implement the routing behavior.	workflow-106-IterativeDesign
Office Integration	Provides a sample in which Microsoft Excel attachments are enabled with workflow notifications.	

24.4 Introduction to Human Workflow Architecture

This section provides an overview of human workflow architecture. The following topics are discussed:

- The services that perform a variety of operations in the life cycle of a task, such as querying tasks for a user, retrieving metadata information related to a task, and so on.
- The two ways to use a human task:
 - Associated with a BPEL process service component
 - Used in standalone mode
- The role of the service engine in the life of a human task

24.4.1 Human Workflow Services

Starting with release 11g, all human task metadata is stored and managed in the Metadata Service (MDS) repository. The workflow service consists of many services that handle various aspects of human interaction with a business process.

[Figure 24–22](#) shows the following workflow service components:

- Task Service:

The task service provides task state management and persistence of tasks. In addition to these services, the task service exposes operations to update a task, complete a task, escalate and reassign tasks, and so on. The task service is used by the Oracle BPM Worklist to retrieve tasks assigned to users. This service also determines if notifications are to be sent to users and groups when the state of the task changes. The task service consists of the following services.

 - Task Routing Service

The task routing service offers services to route, escalate, and reassign the task. The service makes these decisions by interpreting a declarative specification in the form of the routing slip.
 - Task Query Service

The task query service queries tasks for a user based on a variety of search criterion such as keyword, category, status, business process, attribute values, history information of a task, and so on.
 - Task Metadata Service

The task metadata service exposes operations to retrieve metadata information related to a task.
- Identity Service

The identity service is a thin web service layer on top of the Oracle Application Server 11g security infrastructure or any custom user repository. It enables authentication and authorization of users and the lookup of user properties, roles, group memberships, and privileges.
- Notification Service

The notification service delivers notifications with the specified content to the specified user to any of the following channels: email, telephone voice message, IM, and SMS. See [Section 29.2, "Notifications from Human Workflow"](#) for more information.
- User Metadata Service

The user metadata service manages metadata related to workflow users, such as user work queues, preferences, vacations, and delegation rules.

- Runtime Config Service
The runtime config service provides methods for managing metadata used in the task service runtime environment. It principally supports management of task payload flex field mappings.
- Evidence service
The evidence service supports storage and nonrepudiation of digitally-signed workflow tasks.

Figure 24–22 Workflow Services Components

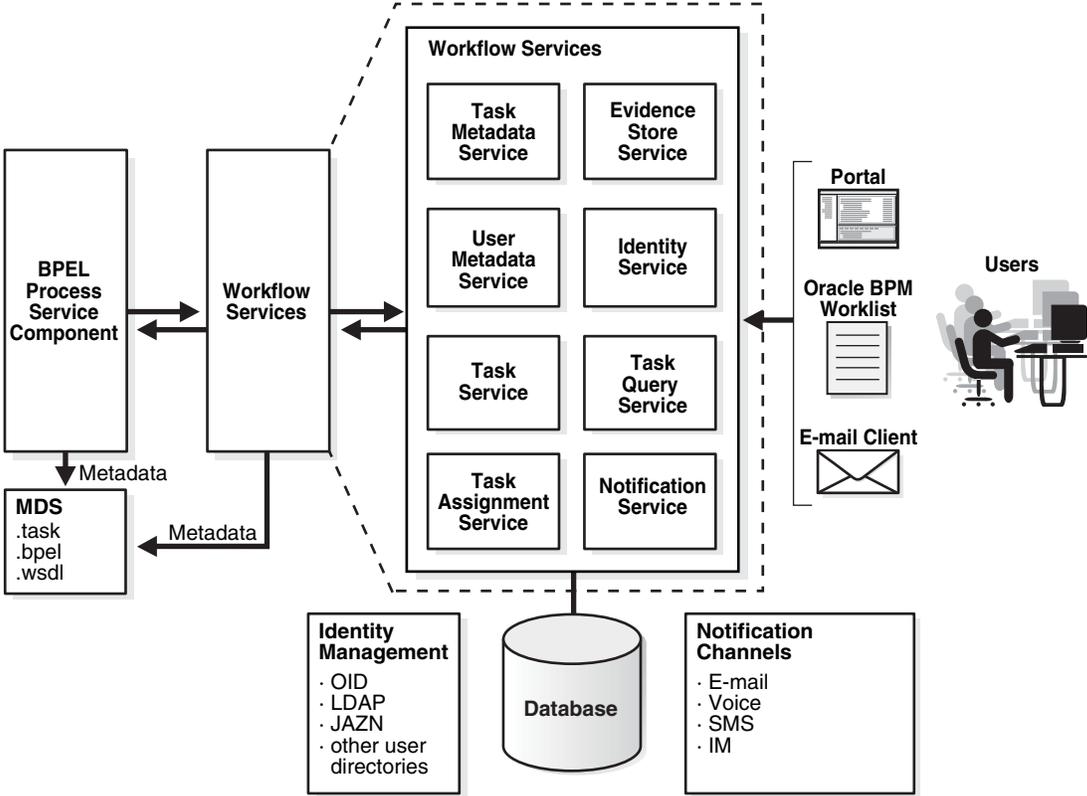
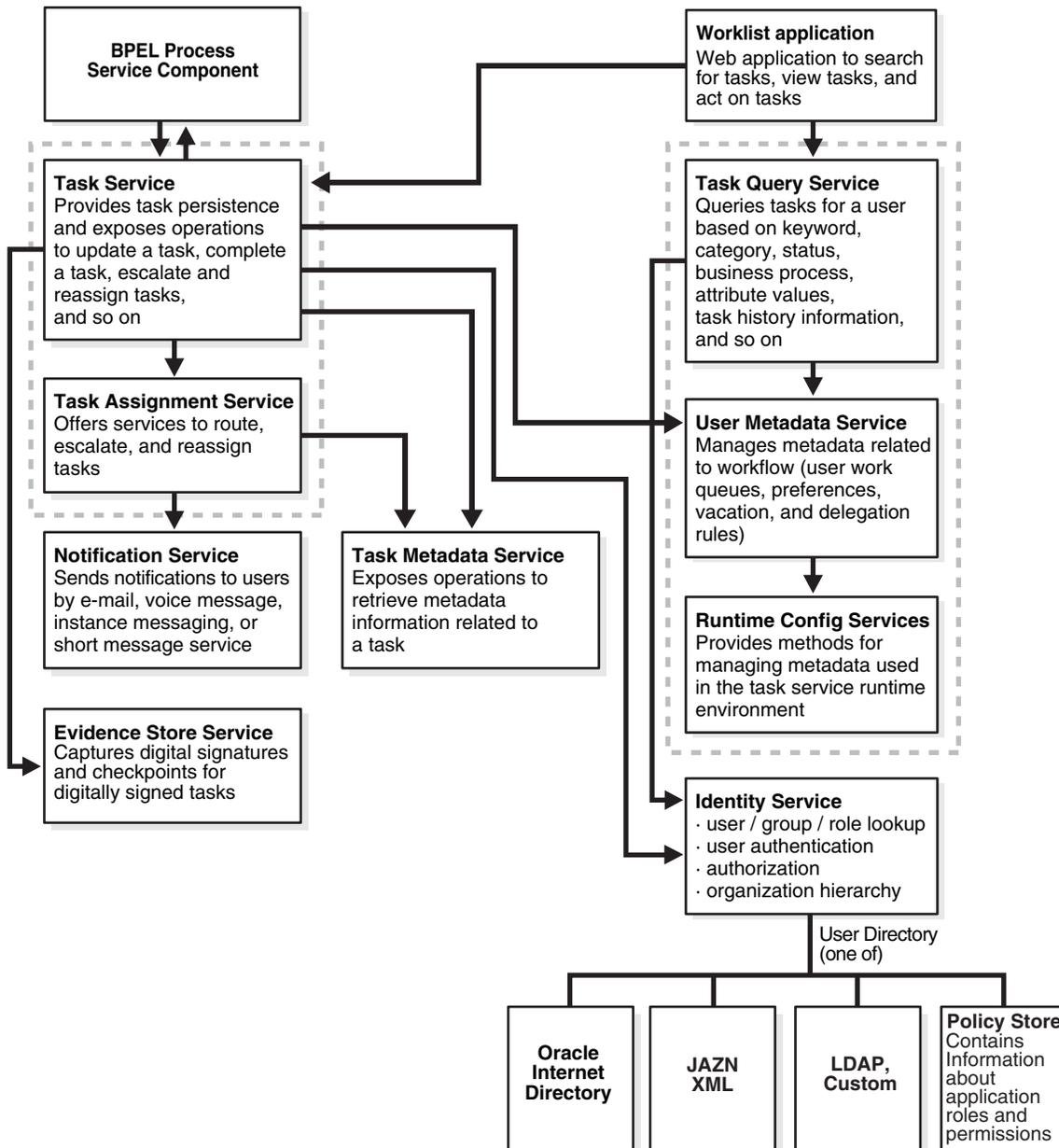


Figure 24–23 shows the interactions between the services and the business process.

Figure 24–23 Workflow Services and Business Process Interactions

24.4.2 Use of Human Task

There are two ways in which to use a human task:

- Human task associated with a BPEL process

In most cases, you associate your human task with a BPEL process. The BPEL process integrates a series of activities (including the human task activity) and services into an end-to-end process flow.

- Standalone human task

You can also create the human task as a standalone component only in the SOA Composite Editor and not associate it with a BPEL process. Standalone human task service components are useful for environments in which there is no need for

any automated activity in an application. In the standalone case, the client can create the task themselves.

24.4.3 Service Engines

During runtime, the business logic and processing rules of the human task service component are executed by the human workflow service engine. Each service component (BPEL process, human workflow, decision service (business rules), and Oracle mediator) has its own service engine container for performing these tasks. All human task service components, regardless of the SOA composite application of which they are a part, are executed in this single human task service engine.

For more information about configuring, monitoring, and managing the human workflow service engine, see *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite*.

Designing Human Tasks

This chapter describes how to design human tasks. It introduces the Human Task Editor to use for modeling task metadata, routing and assignment policies, escalation policies, expiration policies, and notification settings.

This chapter includes the following sections:

- [Section 25.1, "Introduction to Human Task Design Concepts"](#)
- [Section 25.2, "Introduction to the Modeling Process"](#)
- [Section 25.3, "Creating the Human Task Definition with the Human Task Editor"](#)
- [Section 25.4, "Associating the Human Task Service Component with a BPEL Process"](#)

25.1 Introduction to Human Task Design Concepts

To use the Human Task Editor, you must understand human task design concepts, including the following:

- The types of users to which to assign tasks
- The methods by which to assign users to tasks (statically, dynamically, or rule-based)
- The task participant types available for modeling a task to which you assign users
- The options for creating lists of task participants
- The participants involved in the entire life cycle of a task

For information about human task concepts, see [Chapter 24, "Getting Started with Human Workflow."](#)

25.2 Introduction to the Modeling Process

Oracle SOA Suite provides a graphical tool, known as the Human Task Editor, for modeling your task metadata. The modeling process consists of the following:

- Creating and modeling a human task service component in the SOA Composite Editor
- Associating it with a BPEL process
- Generating the task form for displaying the human task during runtime in Oracle BPM Worklist.

This section provides a brief overview of these modeling tasks and provides references to specific modeling instructions.

For more information about using the SOA Composite Editor, see [Chapter 4, "Introduction to the Functionality of the SOA Composite Editor."](#)

For information about available samples, see [Section 24.3.2, "Designing a Human Task from Start to Finish."](#)

25.2.1 Create a Human Task Definition

You define the metadata for the human task in either of two ways:

- By dragging a human task from the Component Palette into a BPEL process and clicking the **Add** icon in the Create Human Task dialog that automatically is displayed. This displays a dialog for creating the human task service component. When creation is complete, the Human Task Editor is displayed.
- By dragging a human task service component from the Component Palette into the SOA Composite Editor. This displays a dialog for creating the human task component. When creation is complete, the Human Task Editor is displayed.

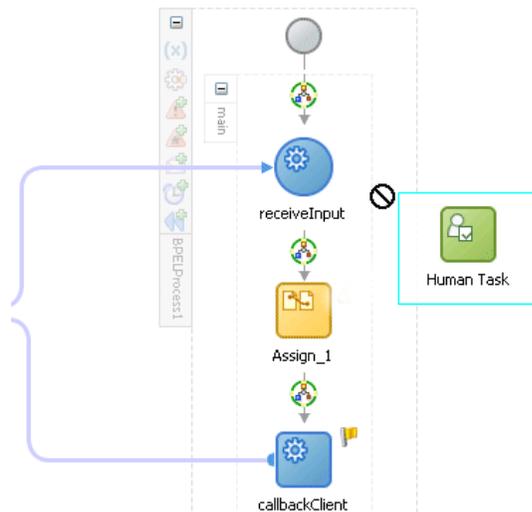
The Human Task Editor enables you to specify human task metadata, such as task outcome, payload structure, assignment and routing policy, expiration and escalation policy, notification settings, and so on. This information is saved to a metadata task configuration file with a `.task` extension. In addition, some workflow patterns may also need to use the Oracle Business Rules Designer to define task routing policies or the list of approvers.

For more information, see [Section 25.3, "Creating the Human Task Definition with the Human Task Editor."](#)

25.2.2 Associate the Human Task Definition with a BPEL Process

You can associate the `.task` file that consists of the human task settings with a BPEL process in Oracle BPEL Designer. Association is made with a human task that you drag into your BPEL process flow for configuring, as shown in [Figure 25–1](#).

Figure 25–1 Dragging a Human Task into a BPEL Process



You also specify the task definition, task initiator, task priority, and task parameter mappings that carry the input data to a BPEL variable. You can also define advanced features, such as the scope and global task variables names (instead of accepting the default names), task owner, identification key, BPEL callback customizations, and whether to extend the human task to include other workflow tasks.

When association is complete, a task service partner link is created. The task service exposes the operations required to act on the task.

You can also create the human task as a standalone component only in the SOA Composite Editor and not associate it with a BPEL process. Standalone human task service components are useful for environments in which there is no need for any automated activity in an application. In the standalone case, the client can create the task themselves.

For more information, see [Section 25.4, "Associating the Human Task Service Component with a BPEL Process."](#)

25.2.3 Generate the Task Display Form

You can generate a task display form using the Oracle Application Development Framework (ADF). This form is used for displaying the task details on which you act at runtime in Oracle BPM Worklist.

For information on generating the task display form, see [Chapter 26, "Designing Task Display Forms for Human Tasks."](#)

25.3 Creating the Human Task Definition with the Human Task Editor

The Human Task Editor enables you to define the metadata for the task. The editor enables you to specify human task settings, such as task outcome, payload structure, assignment and routing policy, expiration and escalation policy, notification settings, and so on.

25.3.1 How to Create a Human Task Service Component

You create a human task service component in the SOA Composite Editor or in Oracle BPEL Designer. After creation, you design the component in the Human Task Editor. The method by which you create the human task service component determines whether the component can be associated later with a BPEL process service component or is a standalone component in the SOA Composite Editor.

To create a human task service component in the SOA Composite Editor:

1. Go to the SOA project in which to create a human task service component in the SOA Composite Editor.
2. From the Component Palette, select **SOA**.
3. From the list, drag a **Human Task** into the designer.
The Create Human Task dialog appears.
4. In the **Name** field, enter a name.
The name you enter is added as the `.task` file name.
5. Note the **Create Composite Service with SOAP Bindings** checkbox. The selection of this checkbox determines how the human task service component is created.

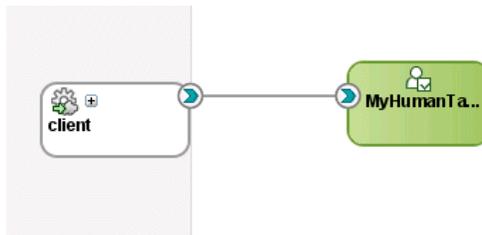
- a. If you want to create a human task service component that you later associate with a BPEL process service component, do *not* select the **Create Composite Service with SOAP Bindings** checkbox. The human task service component is created as a component that you explicitly associate with a BPEL process service component. [Figure 25–2](#) provides details.

Figure 25–2 Human Task Component



- b. If you want to create the human task service component as a standalone component in the SOA Composite Editor, select the **Create Composite Service with SOAP Bindings** checkbox. This creates a human task service component that is automatically wired to a Simple Object Access Protocol (SOAP) web service. [Figure 25–3](#) provides details.

Figure 25–3 Standalone Human Task Component



This web service provides external customers with an entry point into the human task service component of the SOA composite application.

6. Click **Finish**.

To create a human task in Oracle BPEL Designer:

1. From the Component Palette, select **BPEL**.
2. From the list, drag a **Human Task** into the designer.
The Create Human Task dialog appears.
3. Click the **Add** icon to create a human task.
4. In the **Name** field, enter a name.
The name you enter is added as the `.task` file name.
5. In the **Title** field, enter a task.
6. Click **OK**.
The Human Task Editor appears.

Note: You can also create a human task that you *later* associate with a BPEL process by selecting **New** from the **File** main menu, then selecting **SOA Tier > Service Components > Human Task**.

For more information about creating a human task service component in the SOA Composite Editor, see [Chapter 4, "Introduction to the Functionality of the SOA Composite Editor."](#)

25.3.2 What Happens When You Create a Human Task Service Component

When a human task is created, the following folders and files appear:

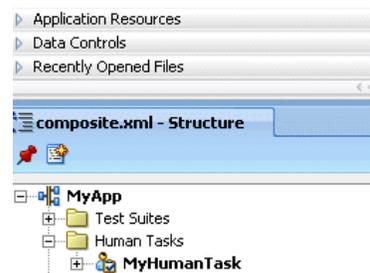
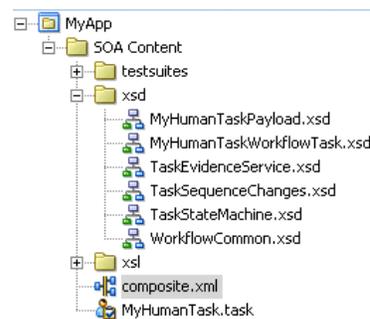
- The human task settings specified in the Human Task Editor are saved to a metadata task configuration file in the metadata service (MDS) repository with a `.task` extension. This file appears in the Application Navigator under **SOA_Project_Name > SOA Content**. You can re-edit the settings in this file by double-clicking the following:
 - The `.task` file in the Application Navigator in either the SOA Composite Editor or Oracle BPEL Designer
 - The human task icon in the SOA Composite Editor or in your BPEL process in Oracle BPEL Designer.

This reopens the `.task` file in the Human Task Editor.

- A **Human Tasks** folder containing the human task you created appears in the Structure window of the SOA Composite Editor.

[Figure 25–4](#) shows these folders and files.

Figure 25–4 Human Task Folders and Files



For information about available samples, see [Section 24.3.2, "Designing a Human Task from Start to Finish."](#)

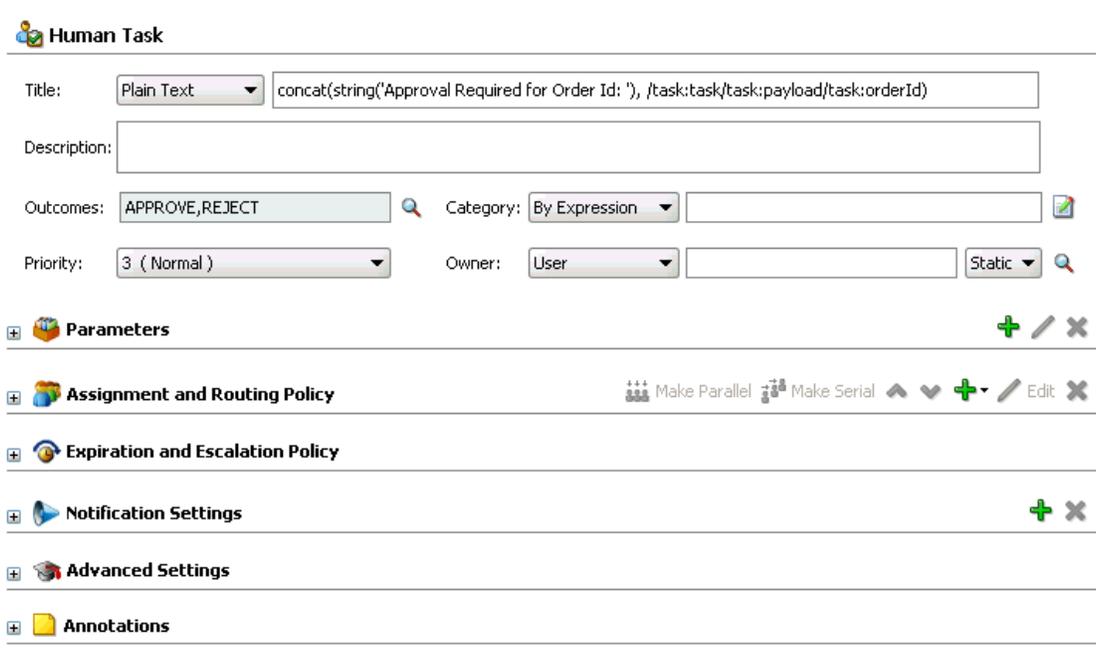
25.3.3 How to Access the Sections of the Human Task Editor

To access the sections of the Human Task Editor:

1. Double-click the **Human Task** icon in the SOA Composite Editor or double-click the Human Task icon in Oracle BPEL Designer and click the **Edit** icon in the upper right corner.

The Human Task Editor consists of the following main sections shown in [Figure 25–5](#). These sections enable you to design the metadata of a human task.

Figure 25–5 Human Task Editor



Instructions for using these main sections of the Human Task Editor to create a workflow task are listed in [Table 25–1](#).

Table 25–1 Human Task Editor

Section	Description	See...
Task Title (title, description, outcomes, category, priority, and owner)	Enables you to define task details such as title, task outcomes, owner, and other attributes.	Section 25.3.4, "How to Specify the Title, Description, Outcome, Priority, Category, and Owner"
Parameters	Enables you to define the structure (message elements) of the task payload (the data in the task).	Section 25.3.5, "How to Specify the Task Payload Data Structure"
Assignment and Routing Policy	Enables you to assign participants to the task and create a policy for routing the task through the workflow.	Section 25.3.6, "How to Assign Task Participants" Section 25.3.7, "How to Select a Routing Policy"

Table 25–1 (Cont.) Human Task Editor

Section	Description	See...
Expiration and Escalation Policy	Enables you to specify the expiration duration of a task	Section 25.3.8, "How to Escalate, Renew, or End the Task"
Notification Settings	Enables you to create and send notifications when a user is assigned a task or informed that the status of the task has changed.	Section 25.3.9, "How to Specify Participant Notification Preferences"
Advanced Settings For specifying:	<p>Enables you to specify advanced settings, such as:</p> <ul style="list-style-type: none"> ▪ Custom escalation rules ▪ WordML and custom style sheets for attachments ▪ Multilingual settings ▪ Callback classes ▪ Workflow signature policies ▪ Access rules to task content ▪ Restrictions on assignments ▪ Task and routing assignments in BPEL callbacks ▪ Graphical history of tasks 	Section 25.3.10, "How To Specify Advanced Settings"
Annotations	Enables you to label different attributes of the task definition. Annotations are used with Oracle Business Process Analysis.	Section 25.3.11, "How to Specify Annotations"

25.3.4 How to Specify the Title, Description, Outcome, Priority, Category, and Owner

Figure 25–6 shows the **Task Title** section of the Human Task Editor.

This section enables you to specify details such as the task title, description, task outcomes, task category, task priority, and task owner.

Figure 25–6 Human Task Editor — Task Title Section

Human Task

Title: Approval Required for Order Id: <math>\langle\%/task:task/task:payload/task:orderId\%>

Description:

Outcomes: Category:

Priority: Owner:

Instructions for configuring the following subsections of the **Task Title** section are listed in Table 25–2:

Table 25–2 Human Task Editor — Task Title Section

For This Subsection...	See...
Title	Section 25.3.4.1, "Specifying a Task Title"
Description	Section 25.3.4.2, "Specifying a Task Description"

Table 25–2 (Cont.) Human Task Editor — Task Title Section

For This Subsection...	See...
Outcomes	Section 25.3.4.3, "Specifying a Task Outcome"
Category	Section 25.3.4.4, "Specifying a Task Category"
Priority	Section 25.3.4.5, "Specifying a Task Priority"
Owner	Section 25.3.4.6, "Specifying a Task Owner"

25.3.4.1 Specifying a Task Title

To specify a task title:

Enter an optional task title. The title defaults to this value only if the initiated task does not have a title set in it. The title provides a visual identifier for the task. The task title displays in Oracle BPM Worklist. You can also search on titles in Oracle BPM Worklist.

1. Select a method for specifying a task title:
 - **Plain Text:** Manually enter a name (for example, `Vacation Request Approved`).
 - **Text and XPath:** Enter a combination of manual text and a dynamic expression. After manually entering a portion of the title (for example, `Approval Required for Order Id:`), place the cursor one blank space to the right of the text and click the icon to the right of this field. This displays the Expression Builder for dynamically creating the remaining portion of the title. After completing the dynamic portion of the name, click **OK** to return to this field. The complete name is displayed. For example:

```
Approval Required for Order Id: <%/task:task/task:payload/task:orderId%>
```

The expression is resolved during runtime with the exact order ID value from the task payload.

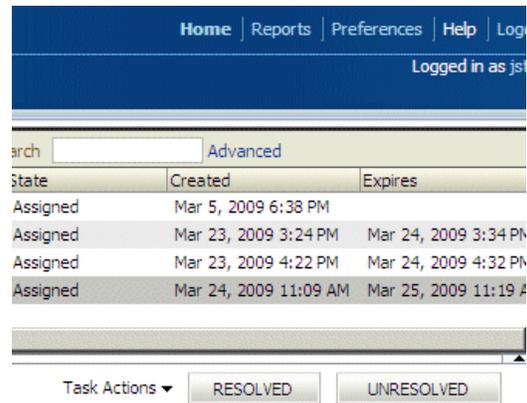
If you enter a title in the **Task Title** field of the **General** tab of the Create Human Task dialog described in [Section 25.4.3.1, "Specifying the Task Title,"](#) the title you enter here is overridden.

25.3.4.2 Specifying a Task Description

You can optionally specify a description of the task in the **Description** field. The description enables you to provide additional details about a task. For example, if the task title is `Computer Upgrade Request`, you can provide additional details in this field, such as the model of the computer, amount of CPU, amount of RAM, and so on. The description does not display in Oracle BPM Worklist.

25.3.4.3 Specifying a Task Outcome

Task outcomes capture the possible outcomes of a task. Oracle BPM Worklist displays the outcomes you specify here as the possible task actions to perform during runtime. [Figure 25–7](#) provides details.

Figure 25–7 Outcomes in Oracle BPM Worklist

You can specify the following types of task outcomes:

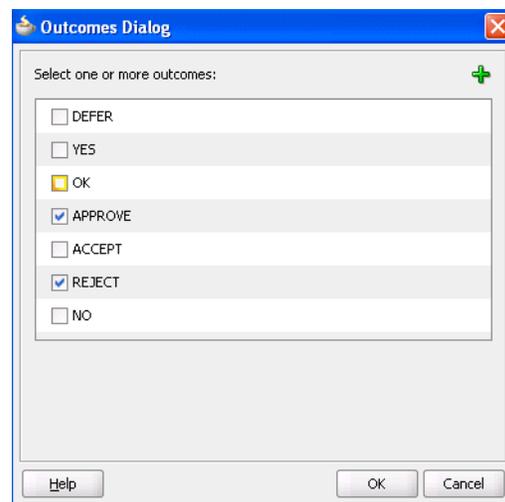
- Select a seeded outcome
- Enter a custom outcome

The task outcomes can also have runtime display values that are different from the actual outcome value specified here. This permits outcomes to be displayed in a different language in Oracle BPM Worklist. For more information about internationalization, see [Section 25.3.10.4, "Specifying Multilingual Settings."](#)

To specify a task outcome:

1. To the right of the **Outcomes** field in the **Task Title** section, click the **Search** icon.

The Outcomes dialog shown in [Figure 25–8](#) displays the possible outcomes for tasks. **APPROVE** and **REJECT** are selected by default.

Figure 25–8 Outcomes Dialog

2. Select additional task outcomes or unselect the default outcomes.
3. To add custom outcomes, click the **Add** icon.
4. In the **Name** field, enter a custom name, and click **OK**.

Note: Ensure that you do not specify a custom name that matches a name listed in the **Task Actions** tab of the Configure Task Content Access dialog (for example, do not specify `Delete`). The Configure Task Content Access dialog is accessible by clicking **Configure Visibility** in the **Advanced Settings** section of the Human Task Editor. Specifying the same name can cause problems at runtime.

5. Click **OK** to return to the Human Task Editor.

Your selections display in the **Outcomes** field.

The seeded and custom outcomes selected here display for selection in the **Majority Voted Outcome** section of the parallel participant type.

For more information, see [Section 25.3.6.2.1, "Specifying the Voting Outcome."](#)

25.3.4.4 Specifying a Task Category

You can optionally specify a task category in the **Category** field. This categorizes tasks created in a system. For example, in a help desk environment, you may categorize customer requests as either software-related or hardware-related. The category displays in Oracle BPM Worklist. You can filter tasks based on category and create views on categories in Oracle BPM Worklist.

To specify a task category:

1. Select a method for specifying a task category:
 - **By Name:** Manually enter a name.
 - **By Expression:** Click the icon to the right of this field to display the Expression Builder for dynamically creating a category.

25.3.4.5 Specifying a Task Priority

Specify the priority of the tasks. Priority can be **1** through **5**, with **1** being the highest. By default, the priority of a task is **3**. This priority value is overridden by any priority value you select in the **General** tab of the Create Human Task dialog. You can filter tasks based on priority and create views on priorities in Oracle BPM Worklist.

To specify a task priority:

1. From the **Priority** list, select a priority for the task.

For more information about specifying a priority value in the Create Human Task dialog, see [Section 25.4.3.2, "Specifying the Task Initiator and Task Priority."](#)

25.3.4.6 Specifying a Task Owner

The task owner can view the tasks belonging to business processes they own and perform operations on behalf of any of the assigned task participant types. Additionally, the owner can also reassign, withdraw, or escalate tasks. The task owner can be considered the business administrator for a task. The task owner can also be specified in the **Advanced** tab of the Create Human Task dialog described in [Section 25.4.4.2, "Specifying a Task Owner."](#) The task owner specified in the **Advanced** tab overrides any task owner you enter here.

For more information about the task owner, see [Section 24.2.1.3, "Task Stakeholders."](#)

To specify a task owner:

1. Select a method for specifying the task owner:
 - Statically through the identity service user directory or the list of application roles
 - Dynamically through an XPath expression

For example:

- If the task has a payload message attribute named `po` within which the owner is stored, you can specify an XPath expression such as:
`/task:task/task:payload/po:purchaseOrder/po:owner`
- `ids:getManager('jstein', 'jazn.com')`

The manager of `jstein` is the task owner.

For more information about users, groups, and application roles, see [Section 24.2.1.1.3, "Participant Assignment."](#)

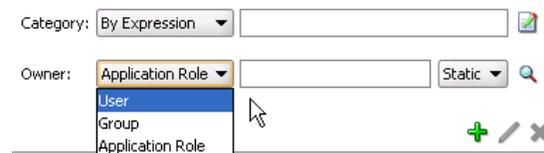
25.3.4.6.1 Specifying a Task Owner Statically Through the User Directory or Application Roles

Task owners can be selected by browsing the user directory (Oracle Internet Directory, Java AuthoriZatioN (JAZN)/XML, LDAP, and so on) or a list of application roles configured for use with Oracle SOA Suite.

To specify a task owner statically through the user directory or a list of application roles:

1. In the first list to the right of the **Owner** field in the **Task Title** section, select **User**, **Group**, or **Application Role** as the type of task owner. [Figure 25–9](#) provides details.

Figure 25–9 Specify a Task Owner By Browsing the User Directory or Application Roles



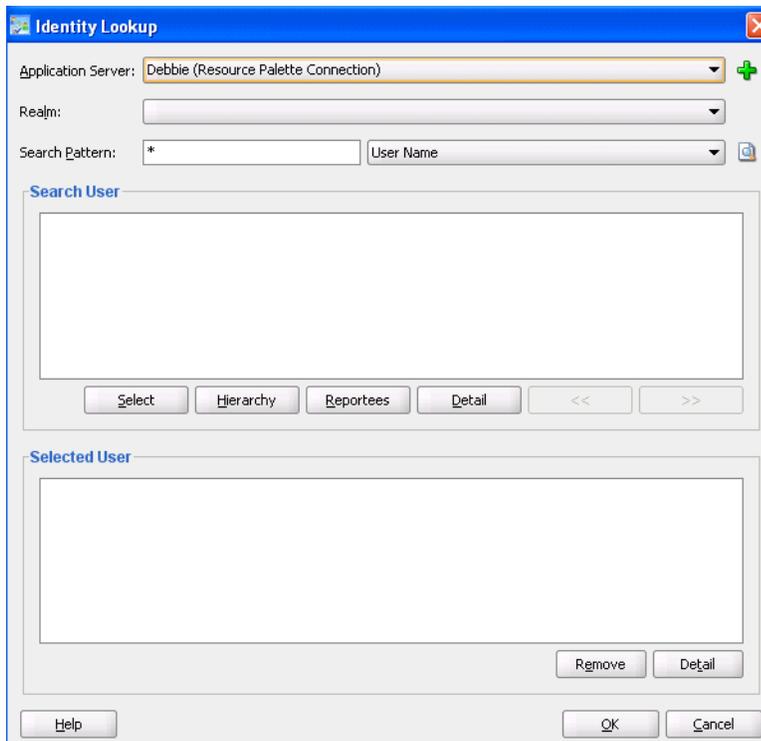
2. In the second list to the right of the **Owner** field in the **Task Title** section, select **Static**.
3. See the step in [Table 25–3](#) based on the type of owner you selected.

Table 25–3 Type of Owner

If You Selected...	See Step...
User or Group	4
Application Role	5

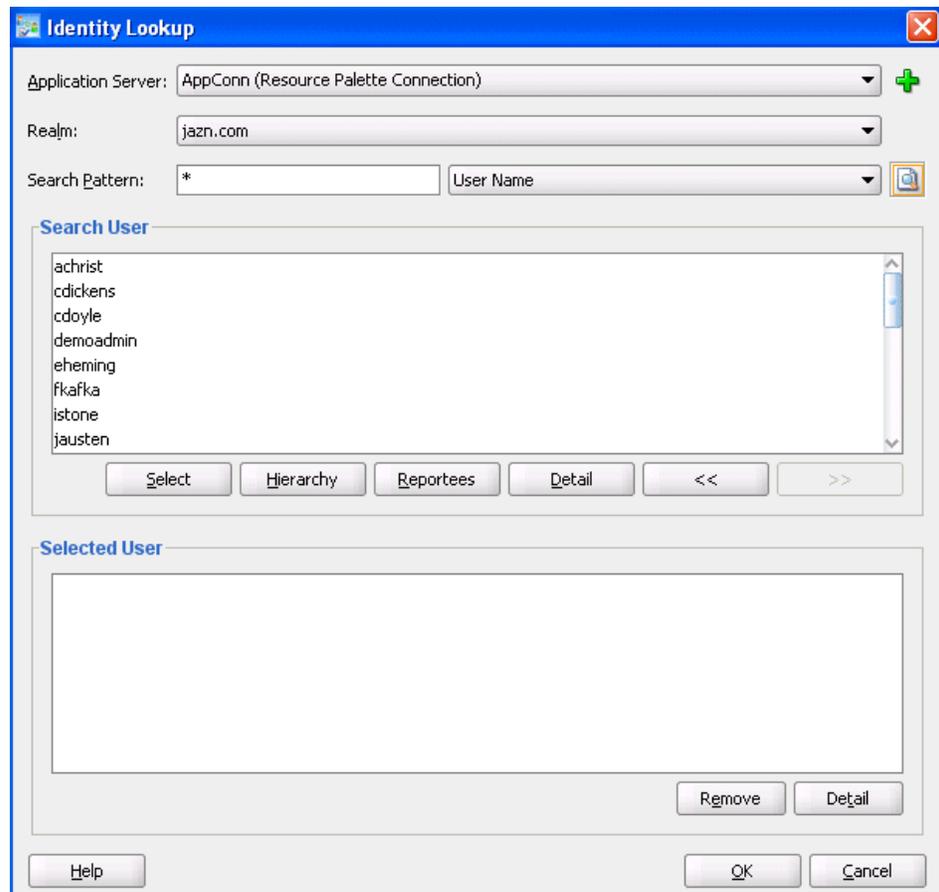
4. If you selected **User** or **Group**, the Identity Lookup dialog shown in [Figure 25–10](#) appears.

Figure 25–10 Identity Lookup Dialog



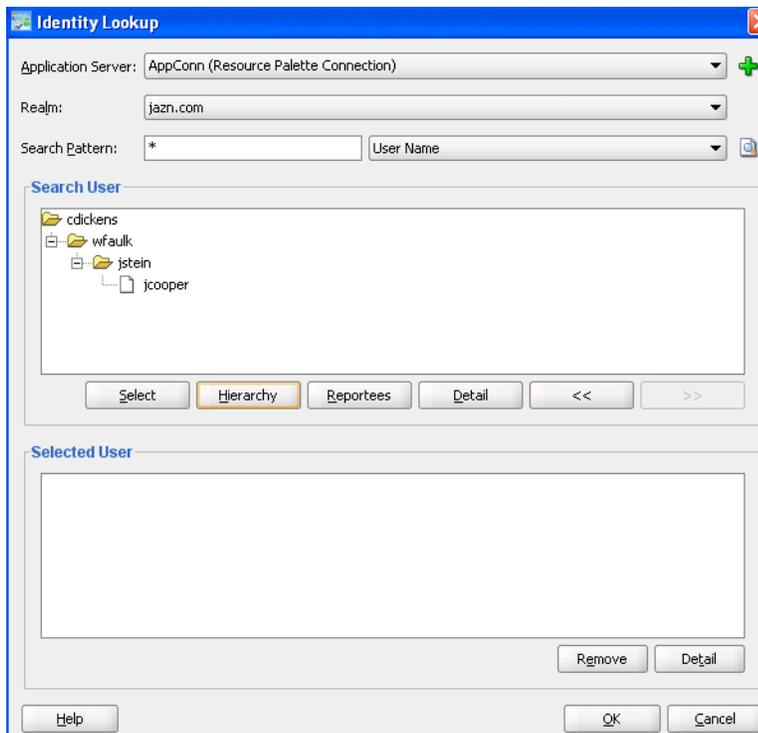
To select a user or group, you must first create an application server connection by clicking the **Add** icon. Note the following restrictions:

- Do not create an application server connection to an Oracle WebLogic Administration Server from which to retrieve the list of identity service realms. This is because there is no identity service running on the Administration Server. Therefore, no realm information displays and no users display when performing a search with a search pattern in the Identity Lookup dialog. Instead, create an application server connection to a managed Oracle WebLogic Server.
- You must select an application server connection configured with the complete domain name (for example, `myhost.us.oracle.com`). If you select a connection configured only with the hostname (for example, `myhost`), the **Realm** list may not display the available realms. If the existing connection does not include the domain name, perform the following steps:
 - In the **Resource Palette**, right-click the application server connection.
 - Select **Properties**.
 - In the **Configuration** tab, add the appropriate domain to the hostname.
 - Return to the Identity Lookup dialog and reselect the connection.
- a. Select or create an application server connection to display the realms for selection. A realm provides access to a policy store of users and roles (groups).
- b. Search for the owner by entering a search string such as `jcooper`, `j*`, `*`, and so on. Clicking the **Lookup** icon to the right of the **User Name** field fetches all the users that match the search criteria. [Figure 25–11](#) provides details. One or more users or groups can be highlighted and selected by clicking **Select**.

Figure 25–11 Identity Lookup with Realm Selected

- c. View the hierarchy of a user by highlighting the user and clicking **Hierarchy**. Similarly, clicking **Reportees** displays the reportees of a selected user or group. [Figure 25–12](#) provides details.

Figure 25–12 User Hierarchy in Identity Lookup Dialog



- d. View the details of a user or group by highlighting the user or group and clicking **Detail**. Figure 25–13 provides details.

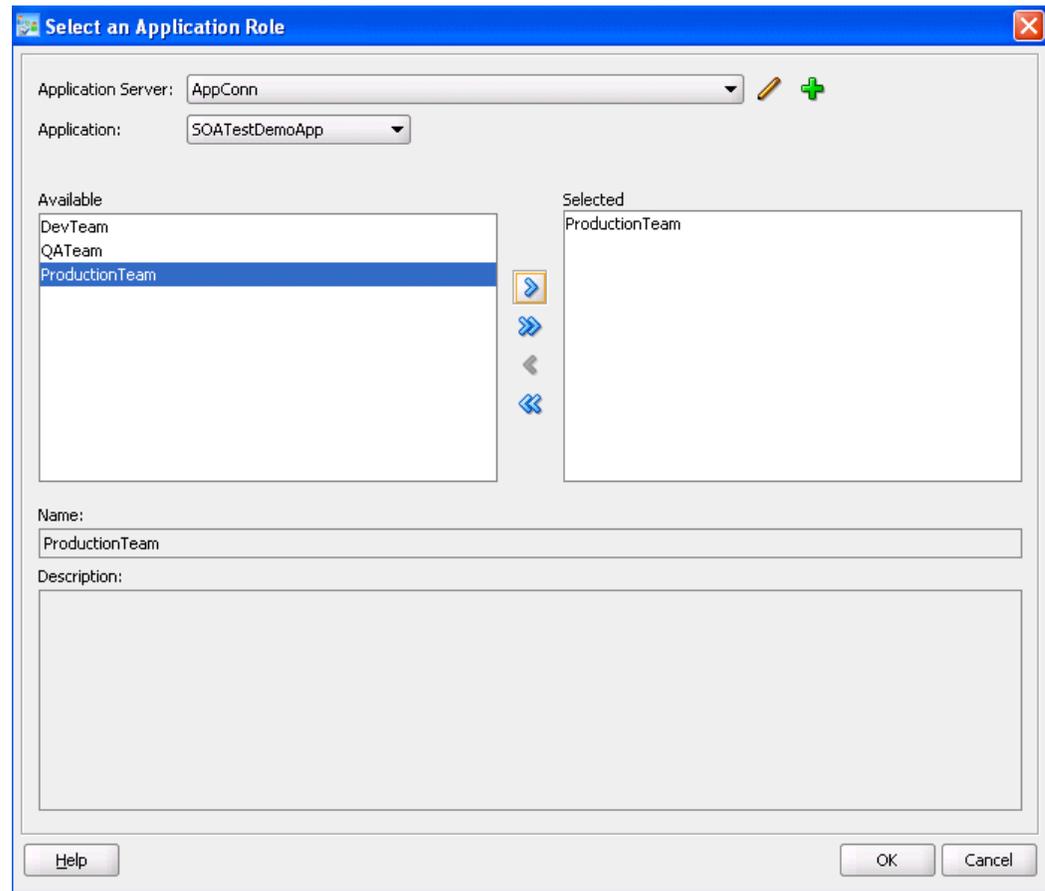
Figure 25–13 User or Group Details



- e. Click **OK** to return to the Identity Lookup dialog.
 - f. Click **Select** to add the user to the **Selected User** section.
 - g. Click **OK** to return to the Human Task Editor.
Your selection displays in the **Owner** field.
5. If you selected **Application Role**, the Select an Application Role dialog appears.
 - a. In the **Application Server** list, select the type of application server that contains the application role or click the **Add** icon to launch the Create Application Server Connection wizard to create a connection.

- b. In the **Application** list, select the application that contains the application roles (for example, a custom application or **soa-infra** for the SOA Infrastructure application).
- c. In the **Available** section, select appropriate application roles and click the > button. To select all, click the >> button. [Figure 25–14](#) provides details.

Figure 25–14 Application Role



- d. Click OK.

25.3.4.6.2 Specifying a Task Owner Dynamically Through an XPath Expression

Task owners can be selected dynamically in the Expression Builder dialog.

To specify a task owner dynamically:

1. In the first list to the right of the **Owner** field in the **Task Title** section, select **User**, **Group**, or **Application Role** as the type of task owner. [Figure 25–15](#) provides details.

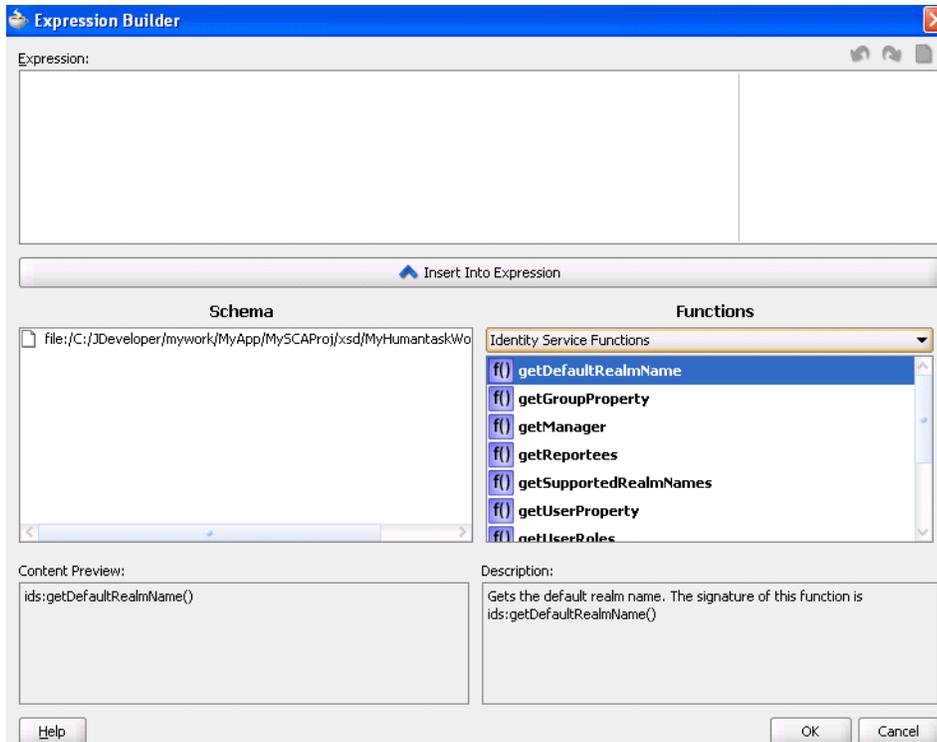
Figure 25–15 Specify a Task Owner Dynamically



- In the second list to the right of the **Owner** field in the **Task Title** section, select **XPath**.

This displays the Expression Builder dialog shown in [Figure 25–16](#):

Figure 25–16 Expression Builder



- Browse the available variable schemas and functions to create a task owner.
- Click **OK** to return to the Human Task Editor.
Your selection displays in the **Owner** field.
For more information, see the following:
 - Click **Help** for instructions on using the Expression Builder dialog and XPath Building Assistant
 - [Appendix B, "XPath Extension Functions"](#) for information about workflow service dynamic assignment functions and identity service functions

25.3.5 How to Specify the Task Payload Data Structure

[Figure 25–17](#) shows the **Parameters** section of the Human Task Editor.

This section enables you to specify the structure (message elements) of the task payload (the data in the task) defined in the XSD file. You create parameters to represent the elements in the XSD file. This makes the payload data available to the workflow task. For example:

- You create a parameter for an order ID element for placing an order from a store front application
- You create parameters for the location, type, problem description, severity, status, and resolution elements for creating a help desk request

Task payload data consists of one or more elements or types. Based on your selections, an XML schema definition is created for the task payload.

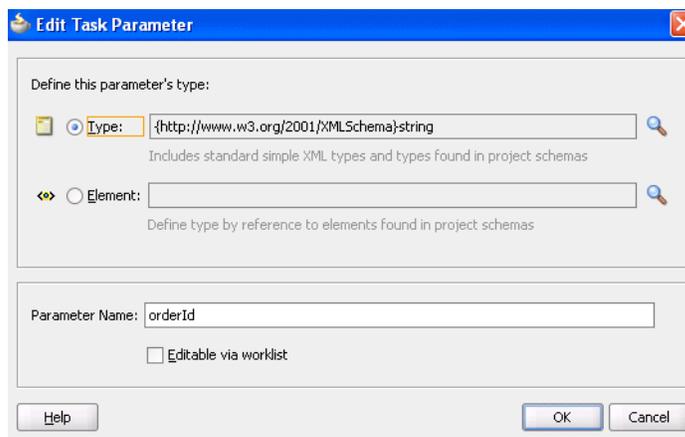
Figure 25–17 Human Task Editor — Parameters Section



To specify the task payload data structure:

1. In the **Parameters** section, click the **Add** icon to display the Add Task Parameter dialog shown in [Figure 25–18](#).

Figure 25–18 Add Task Parameter Dialog



2. Enter the details described in [Table 25–4](#):

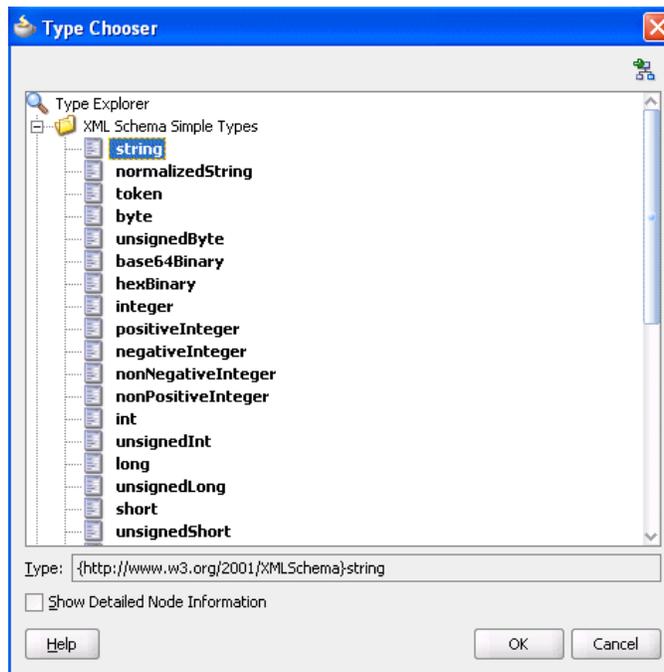
Table 25–4 Add Task Parameter Dialog Fields and Values

Field	Description
Parameter Type	Select Type or Element and click the Search icon to display the Type Chooser dialog for selecting the task parameter.
Parameter Name	Accept the default name or enter a custom name. This field only displays if Type is the selected parameter type.
Editable via worklist	Select this checkbox to enable users to edit this part of the task payload in Oracle BPM Worklist. For example, for a loan approval task, the APR attribute may need to be updated by the user reviewing the task, but the SSN field may not be editable. You can also specify access rules that determine the parts of a task that participants can view and update. For more information, see Section 25.3.10.8, "Specifying Access Policies on Task Content."

Note: You can only define payload flex field mappings in Oracle BPM Worklist for payload parameters that are simple XML types (string, integer, and so on) or complex types (for example, a purchase order, and so on). If you must search tasks using keywords or define views or delegation rules based on task content, then you must use payload parameters based on simple XML types. These simple types can be mapped to flex columns in Oracle BPM Worklist.

3. Select the type, as shown in [Figure 25–19](#).

Figure 25–19 *Parameter Type*

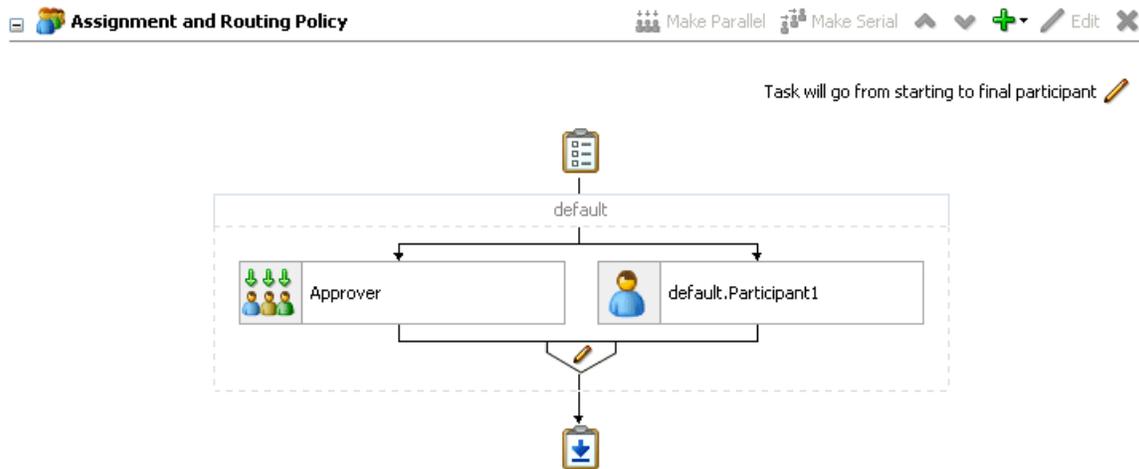


4. Click **OK** to return to the Human Task Editor.
Your selection displays in the **Parameters** section.
5. If you want to edit your selection, select it and click the **Edit** icon in the upper right part of the **Parameters** section.

25.3.6 How to Assign Task Participants

[Figure 25–20](#) shows the **Assignment and Routing Policy** section of the Human Task Editor. This section enables you to select a participant type that meets your business requirement. While configuring the participant type, you build lists of users, groups, and application roles to act upon tasks.

Figure 25–20 Human Task Editor — Assignment and Routing Policy Section



You can easily mix and match participant types to create simple or complex workflow routing policies. You can also extend the functionality of a previously configured human task to model more complex workflows.

A participant type is grouped in a block under a stage (for example, named **default.Participant1** in Figure 25–20). A stage is a way of organizing the approval process for blocks of participant types. You can have one or more stages in sequence or in parallel. Within each stage, you can have one or more participant type blocks in sequence or in parallel. The up and down keys enable you to rearrange the order of your participant type blocks.

For example:

- You can create all participant type blocks in a single stage (for example, a purchase order request in which the entire contents of the order are approved or rejected as a whole).
- You can create more complex approval tasks that may include one or more stages. For example, you can place one group of participant type blocks in one stage and another block in a second stage. The list of approvers in the first stage handles line entry approvals and the list of approvers in the second stage handles header entry approvals.

Each of the participant types has an associated editor that you use for configuration tasks. The sequence in which the assignees are added indicates the execution sequence.

If you want to specify a different stage name or have a business requirement that requires you to create additional stages, perform the following steps. Note that creating additional stages is an advanced requirement that may not be necessary for your environment.

For more information about participant types, see [Section 24.2.1.1, "Task Assignment and Routing."](#)

To specify a stage name and add parallel and sequential blocks:

The stage is named **default** by default. If you want, you can change the name.

1. Double-click the name.

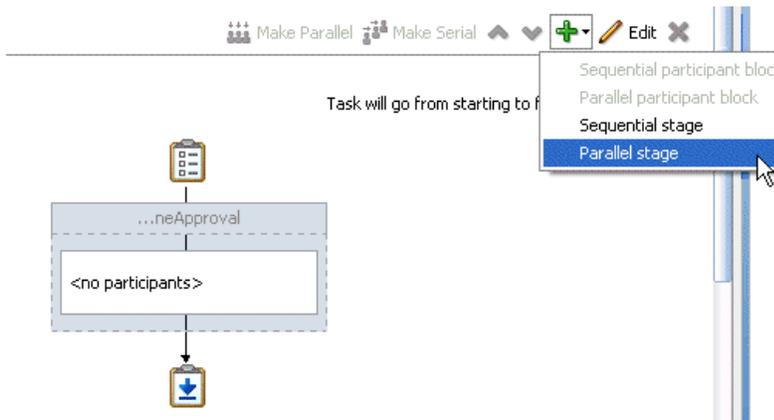
The Edit dialog shown in [Figure 25–21](#) appears.

Figure 25–21 Edit Dialog



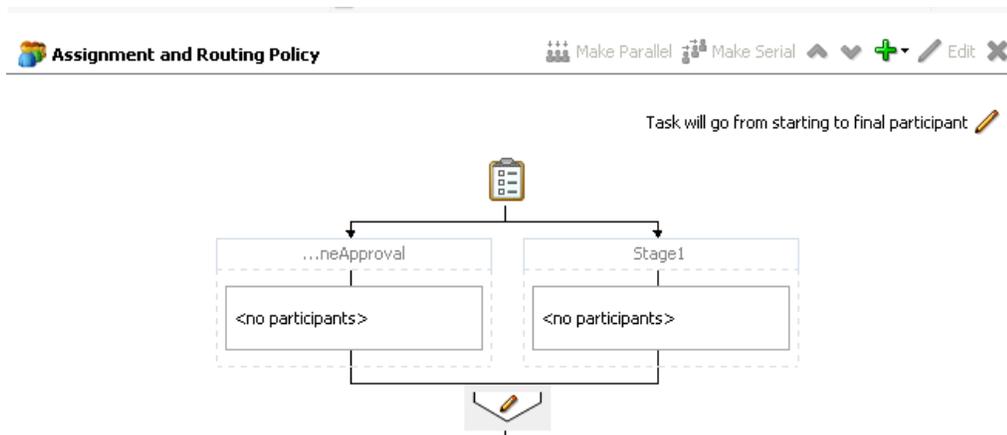
2. Enter a name, and click **OK**.
3. Highlight the stage and its participant type block, and click the **Add** icon. [Figure 25–23](#) shows the list that is displayed.

Figure 25–22 Add a Second Stage



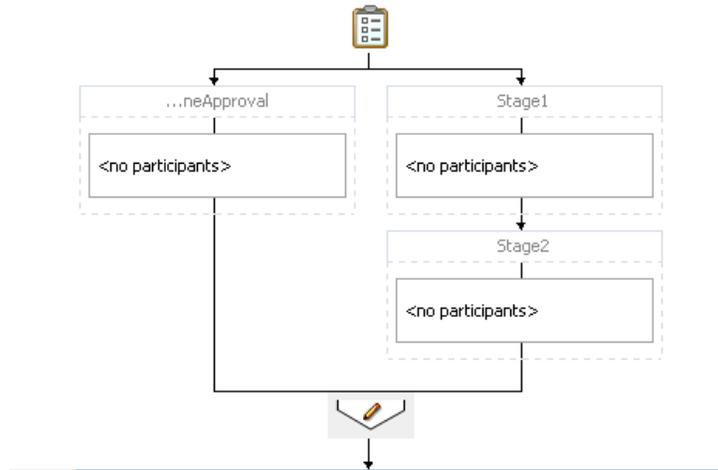
4. Select an option from the list (for example, **Parallel stage**).
A second stage is added in parallel to the first stage, as shown in [Figure 25–23](#).

Figure 25–23 Parallel Stage



5. Highlight the second block on the right, and select the **Add** icon.
6. Select **Sequential stage**.
A sequential stage is added below the selected block.

Figure 25–24 Sequential Stage



You create participant types within these blocks.

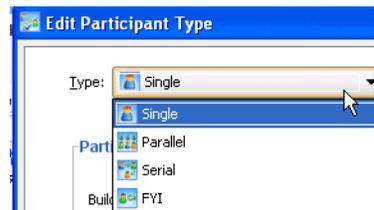
To assign task participants:

1. In the **Assignment and Routing Policy** section, perform one of the following tasks:
 - a. Highlight the block below the stage box and click the **Edit** icon in the upper right corner. The first time you create a task participant, the box is labeled **<no participants>**.
 - or
 - b. Double-click the participant box below the stage box.

The Edit Participant Type dialog appears. This dialog enables you to select a specific participant type.

2. From the **Type** list, select a participant type shown in [Figure 25–25](#).

Figure 25–25 Type List



3. See the section shown in [Table 25–5](#) based on your selection.

Table 25–5 Participant Types

Participant Type	For a Description of this Participant Type, See...	For Instructions on Configuring this Participant Type, See...
■ Single	Section 24.2.1.1.2, "Participant Type"	Section 25.3.6.1, "Configuring the Single Participant Type"
■ Parallel		Section 25.3.6.2, "Configuring the Parallel Participant Type"
■ Serial		Section 25.3.6.3, "Configuring the Serial Participant Type"
■ FYI		Section 25.3.6.4, "Configuring the FYI Participant Type"

25.3.6.1 Configuring the Single Participant Type

Figure 25–26 displays the Edit Participant Type dialog for the single participant type.

Figure 25–26 Edit Participant Type — Single Type

Type: Single Label: Stage1.Participant1
e.g., Approval Manager

Participant List

Build a list of participants using: Names and expressions

Specify attributes using: Value-based Rule-based

Participant Names	Identification Type	Data Type	Value

Requires action from one of the specified participants

Advanced ⬆

Limit allocated duration to:

Day 0 Hour 0 Minutes 0

Over all task duration is currently set to {0} days, {1} hours, and {2} minutes.

Allow this participant to invite other participants

Specify skip rule

To configure the single participant type:

1. In the **Label** field, enter a recognizable label for this participant. This label must be unique among all the participants in the task definition (for example, `Approval Manager`, `Primary Reviewers`, and so on).

Instructions for configuring the following subsections of the Edit Participant Type dialog for the single participant type are listed in [Table 25–6](#):

Table 25–6 Edit Participant Type — Single Type

For This Subsection...	See...
Participant List	Section 25.3.6.1.1, "Creating a Single Task Participant List"
Limit allocated duration to (under the Advanced section)	Section 25.3.6.1.2, "Specifying a Time Limit for Acting on a Task"

Table 25–6 (Cont.) Edit Participant Type — Single Type

For This Subsection...	See...
Allow this participant to invite other participants (under the Advanced section)	Section 25.3.6.1.3, "Inviting Additional Participants to a Task"
Specify skip rule (under the Advanced section)	Section 25.3.6.1.4, "Bypassing a Task Participant"

25.3.6.1.1 Creating a Single Task Participant List

Users assigned to the list of participants can act upon tasks. In this type of assignment list, only one user is required to act on the task. You can provide either a single user or a list of users, groups, or application roles for this pattern. If a list is specified, then all users are assigned the task; one of them must acquire and act upon the task. When one user acts on it, the task is withdrawn from the task list of other assignees.

You can create several types of lists for the single user participant (and also for the parallel, serial, and FYI user participants):

- Value-based name and expression lists

These lists enable you to statically or dynamically select users, groups, or application roles as task assignees.

- Value-based management chain lists

Management chains are typically used for serial approvals through multiple users in a management chain hierarchy. Therefore, this list is most likely useful with the serial participant type. This is typically the case if you want all users in the hierarchy to act upon the task. Management chains can also be used with the single participant type. In this case, however, all users in the hierarchy get the task assigned at the same time. As soon as one user acts on the task, it is withdrawn from the other users.

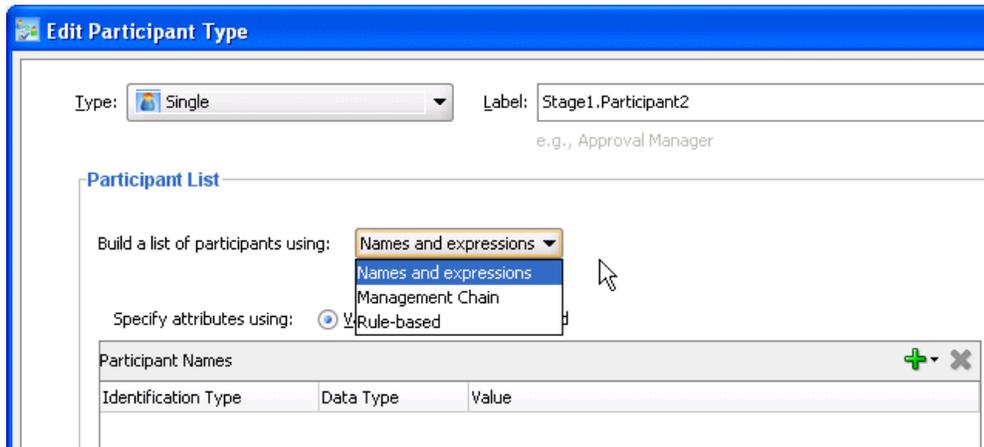
For example, a purchase order is assigned to a manager. If the manager approves the order, it is assigned to their manager. If that manager approves it, it is assigned to their manager, and so on until three managers approve the order. If any managers reject the request or the request expires, the order is rejected if you specify an abrupt termination condition. Otherwise, the task flow continues to be routed.

- Rule-based names and expression lists and management chain lists

Business rules enable you to create the list of task participants with complex expressions. For example, you create a business rule in which a purchase order request below \$5000 is sent to a manager for approval. However, if the purchase order request exceeds \$5000, the request is sent to the manager of the manager for approval. Two key features of business rules are facts and action types, which are described in [Section 25.3.7.2, "Specifying Advanced Task Routing Using Business Rules."](#)

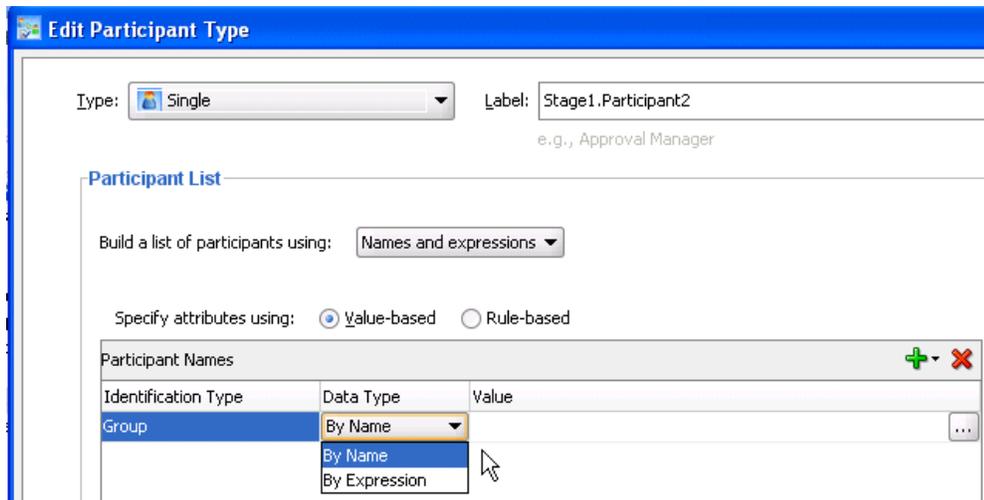
When you select a participant type, the dialog that displays enables you to choose an option for building your list of task participant assignees (users, groups, or application roles), as shown in [Figure 25–27](#). The three selections described above are available: **Names and expressions**, **Management Chain**, and **Rule-based**.

Figure 25–27 Build a List of Participants



After selecting an option, you dynamically assign task participant assignees (users, groups, or application roles) and a data type, as shown in [Figure 25–28](#).

Figure 25–28 Assignment of Task Assignees



This section describes how to create these lists of participants.

Creating Participant Lists Consisting of Value-Based Names and Expressions

Select a method for statically or dynamically assigning a user, group, or application role as a task participant.

For information about the following:

- Users, groups, or application roles, see [Section 24.2.1.1.3, "Participant Assignment."](#)
- Statically and dynamically assigning task participants, see [Section 24.2.1.2, "Static, Dynamic, and Rule-Based Task Assignment."](#)

To create participant lists consisting of value-based names and expressions:

1. From the **Build a list of participants using** list, select **Names and expressions**.
2. From the **Specify attributes using** list, select **Value-based**.

The dialog refreshes to display the fields shown in [Figure 25–29](#).

Figure 25–29 Value-Based Names and Expressions

Participant List

Build a list of participants using: Names and expressions ▼

Specify attributes using: Value-based Rule-based

Identification Type	Data Type	Value
User	By Name	jstein

Requires action from one of the specified participants

3. Click the **Add** icon and select a user, group, or application role as a task participant.

The **Identification Type** column of the **Participant Names** table displays your selection of user, group, or application role.

4. If you want to change your selection in the **Identification Type** column, click it to invoke a dropdown list.
5. In the **Data Type** column, click your selection to invoke a dropdown list to assign a value:

- **By Name:** If your identification type is a user or group, click the **Browse** icon (the dots) on the right to display a dialog for selecting a user or group configured through the identity service. The identity service enables the lookup of user properties, roles, and group memberships. User information is obtained from an LDAP server such as Oracle Internet Directory. You can use wild cards (*) to search for IDs.

If your selection is an application role, click the **Browse** icon to display the Select an Application Role dialog for selecting an application role. To search for application roles, you must first create a connection to the application server. When searching, you must specify the application name to find the name of the role. Note that the task definition can refer to only one application name. You cannot use application roles from different applications as assignees or task owners.

- **By Expression:** For a user, group, or application role, click the **Browse** icon to dynamically select a task assignee in the Expression Builder dialog. Use the `bpws:getVariableData(...)` expression or the `ids:getManager()` XPath function.

The **Value** column displays the value you specified.

6. If you want to manually enter a value, click the field in the **Value** column and specify a value.

Creating Participant Lists Consisting of Value-Based Management Chains

Select a method for statically or dynamically assigning management chain parameters as task participants.

For information about the following:

- Users, groups, or application roles, see [Section 24.2.1.1.3, "Participant Assignment."](#)
- Statically and dynamically assigning task participants, see [Section 24.2.1.2, "Static, Dynamic, and Rule-Based Task Assignment."](#)
- Management chains, see [Section 25.3.6.1.1, "Creating a Single Task Participant List."](#)

To specify participant lists based on value-based management chains:

1. From the **Build a list of participants using** list, select **Management Chain**.
2. From the **Specify attributes using** list, select **Value-based**.

The dialog refreshes to display the fields shown in [Figure 25–30](#).

Figure 25–30 Value-Based Management Chains

Participant List

Build a list of participants using: Management Chain

Specify attributes using: Value-based Rule-based

Starting Participant:		
Identification Type	Data Type	Value

Top Participant: By Title

Number of Levels: By Number

Management Chain list builder stops when Top Participant is reached or number of levels is met

3. See Step 3 through Step 6 on page 25-25 for instructions on assigning a user, group, or application role to a list in the **Starting Participant** table.
4. In the **Top Participant** list, select a method for assigning the number of task participant levels:
 - **By Title:** Select the title of the last (highest) approver in the management chain.
 - **XPath:** Select to dynamically enter a top participant through the Expression Builder dialog.
5. In the **Number of Levels** list, select a method for assigning a top participant:
 - **By Number:** Enter a value for the number of levels in the management chain to include in this task. For example, if you enter 2 and the task is initially

assigned to user `jcooper`, both the user `jstein` (manager of `jcooper`) and the user `wfaulk` (manager of `jstein`) are included in the list (apart from `jcooper`, the initial assignee).

- **XPath:** Select to dynamically enter a value through the Expression Builder dialog.

Creating Participant Lists Consisting of Rulesets

A ruleset provides a unit of execution for rules and for decision tables. In addition, rulesets provide a unit of sharing for rules; rules belong to a ruleset. Multiple rulesets can be executed in order. This is called rule flow. The ruleset stack determines the order. The order can be manipulated by rule actions that push and pop rulesets on the stack. In rulesets, the priority of rules applies to specify the order of firing of rules in the ruleset. Rulesets also provide an effective date specification that identifies that the ruleset is always active, or that the ruleset is restricted based on a time and date range, or a starting or ending time and date.

The method by which you create a ruleset is based on how you access it. This is described in the following section.

To specify participant lists based on rulesets:

Business rules can define the participant list. There are two options for using business rules:

- Rules define parameters of a specific list builder (such as **Names and Expressions** or **Management Chain**). In this case, the task routing pattern is modeled to use a specific list builder. In the list builder, the parameters are listed as coming from rules. Rules return the list builder of the same type as the one modeled in Oracle JDeveloper.
 1. From the **Build a list of participants using** list, select **Names and expressions** or **Management Chain**.
 2. From the **Specify attributes using** list, select **Rule-based**.
 3. In the **List Ruleset** field, enter a ruleset name.

[Figure 25–31](#) provides details.

Figure 25–31 Rulesets

The screenshot shows a configuration window for a participant list. At the top, there are two fields: 'Type' with a dropdown menu showing 'Single' and 'Label' with a text box containing 'Stage1.Participant1'. Below the 'Label' field is a small example text: 'e.g., Approval Manager'. The main section is titled 'Participant List' and contains three rows of configuration options:

- 'Build a list of participants using:' with a dropdown menu showing 'Names and expressions'.
- 'Specify attributes using:' with two radio buttons: 'Value-based' (unselected) and 'Rule-based' (selected).
- 'List Ruleset:' with a text box containing 'ApprovalGroupRule'.

4. Click **OK**.
- Rules define the list builder and the list builder parameters. In this case, the list itself is built using rules. The rules define the list builder and the parameters.

1. From the **Build a list of participants using** list, select **Rule-based**.
2. In the **List Ruleset** field, enter a ruleset name.

Figure 25–32 provides details.

Figure 25–32 Rulesets

3. Click **OK**.

Both options create a rule dictionary if one is not created and several rule functions and facts are preseeded for easy specifications of the participant list. In the rule dictionary, the following rule functions are seeded to create participant lists:

- CreateResourceList
- CreateManagementChainList

The Task fact is asserted by the task service for basing rule conditions.

After the rule dictionary is created, the Oracle Business Rules Designer is displayed.

1. Model your rule conditions. In the action part, call one of the above functions to complete building your lists. Figure 25–33 provides details.

Figure 25–33 Business Rules

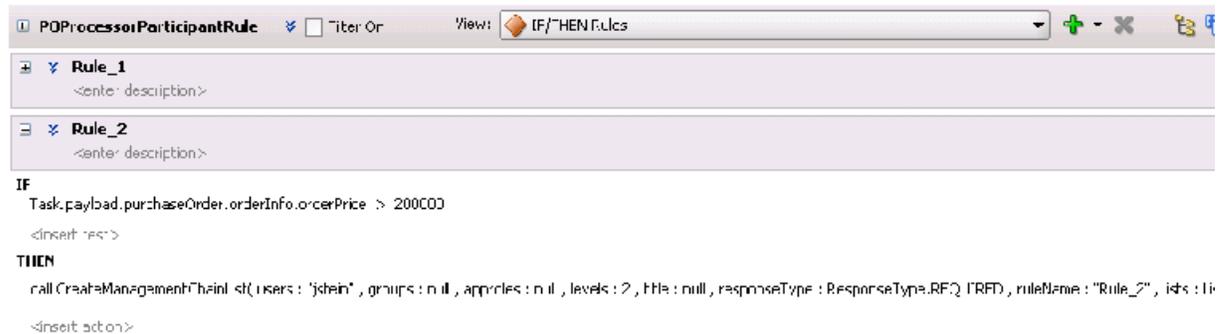


The parameters for the rule functions are similar to the ones in Oracle JDeveloper modeling. In addition to the configurations in Oracle JDeveloper, some additional options are available in the Oracle Business Rules Designer for the following attributes:

- **responseType**: If the response type is **REQUIRED**, the assignee must act on the task. Otherwise, the assignment is converted to an FYI assignment.
- **ruleName**: The rule name can be used to create reasons for assignments.
- **lists**: This object is a holder for the lists that are built. Clicking this option shows a pre-asserted fact **Lists** object to use as the parameter.

An example of rules specifying management chain-based participants is shown in [Figure 25–34](#).

Figure 25–34 Business Rules



If multiple rules are fired, the list builder created by the rule with the highest priority is selected.

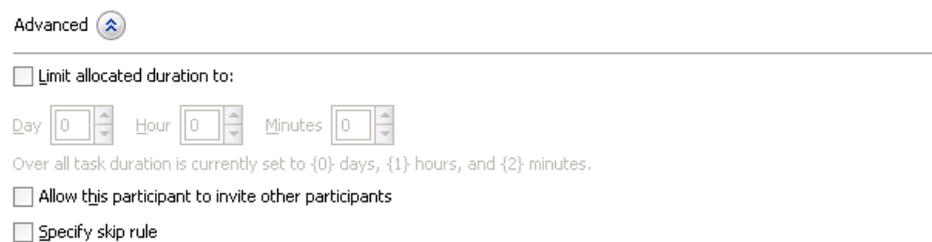
25.3.6.1.2 Specifying a Time Limit for Acting on a Task

You can specify the amount of time a user, group, or application role receives to act on a task. If the user, group, or role does not act in the time specified, the global escalation and renewal policies that you set in the **Expiration and Escalation Policy** section (known as the routing slip level) of the Human Task Editor are applied. For example, if the global policy is set to escalate the task and this participant does not act in the duration provided, the task is escalated to the manager or another user, as appropriate.

To specify a time limit for acting on a task:

1. Expand the **Advanced** section of the Edit Participant Type dialog for the single type, as shown in [Figure 25–35](#).

Figure 25–35 Advanced Section of Edit Participant Type — Single Type



2. Select **Limit allocated duration to**.
3. Specify the amount of time.

For more information about setting the global escalation and renewal policies in the **Expiration and Escalation Policy** section of the Human Task Editor, see [Section 25.3.8, "How to Escalate, Renew, or End the Task."](#)

25.3.6.1.3 Inviting Additional Participants to a Task

You can allow a task assignee to invite other participants into the workflow before routing it to the next assignee in this workflow. For example, assume the approval

workflow goes from James Cooper to John Steinbeck. If this option is checked, James Cooper can decide to first route it to Irving Stone before it goes to John Steinbeck.

This is also known as ad hoc routing. If this option is selected, **Adhoc Route** is added to the **Actions** list in Oracle BPM Worklist at runtime.

To invite additional participants to a task:

1. Expand the **Advanced** section of the Edit Participant Type dialog for the single type, as shown in [Figure 25–35](#).
2. Select **Allow this participant to invite other participants**.

25.3.6.1.4 Bypassing a Task Participant

You can bypass a task participant (user, group, or application role) if a specific condition is satisfied. For example, if a user submits a business trip expense report that is under a specific amount, no approval is required by their manager.

To bypass a task:

1. Expand the **Advanced** section of the Edit Participant Type dialog for the single type, as shown in [Figure 25–35](#).
2. Select **Specify skip rule**.

This action displays an icon for accessing the Expression Builder dialog for building a condition.

The expression to bypass a task participant must evaluate to a boolean value. For example, `/task:task/task:payload/order:orderAmount < 1000` is a valid XPath expression for skipping a participant.

For more information about creating dynamic rule conditions, see [Section 25.3.7.2, "Specifying Advanced Task Routing Using Business Rules."](#)

25.3.6.2 Configuring the Parallel Participant Type

[Figure 25–36](#) and [Figure 25–37](#) display the upper and lower sections of the Parallel dialog.

This participant type is used when multiple users, working in parallel, must act simultaneously, such as in a hiring situation when multiple users vote to hire or reject an applicant. You specify the voting percentage that is needed for the outcome to take effect, such as a majority vote or a unanimous vote.

For example, a business process collects the feedback from all interviewers in the hiring process, consolidates it, and assigns a hire or reject request to each of the interviewers. At the end, the candidate is hired if the majority of interviewers vote for hiring instead of rejecting.

Figure 25–36 Edit Participant Type — Parallel Type (Upper Section of Dialog)

Type: Parallel Label:
e.g., Approval Manager

Vote Outcome

Default Outcome:

Consensus Percentage: By Number Minimum to override default outcome

Immediately trigger voted outcome when minimum percentage is met
 Wait until all votes are in before triggering outcome

Participant List

Build a list of participants using: Names and expressions

Specify attributes using: Value-based Rule-based

Participant Names + - x		
Identification Type	Data Type	Value

Assigns the task, in parallel, to each of the specified participants

Figure 25–37 Edit Participant Type — Parallel Type (Lower Section of Dialog)

Share attachments and comments

Advanced ⬆

Limit allocated duration to:

Day Hour Minutes

Overall task will never expire.

Allow this participant to invite other participants

Specify skip rule



Skip group vote if condition is satisfied.

To assign participants to the parallel participant type:

1. In the **Label** field, enter a recognizable label for this participant. This label must be unique among all the participants in the task definition (for example, `Approval Manager`, `Primary Reviewers`, and so on).

Instructions for configuring the following subsections of the Edit Participant Type dialog for the parallel participant type are listed in [Table 25–7](#):

Table 25–7 Edit Participant Type — Parallel Type

For This Subsection...	See...
Default Outcome	Section 25.3.6.2.1, "Specifying the Voting Outcome"
Consensus Percentage	
Immediately trigger voted outcome when minimum percentage is met	
Wait until all votes are in before triggering outcome	
Participant List	Section 25.3.6.2.2, "Creating a Parallel Task Participant List"
Share attachments and comments	Section 25.3.6.2.3, "Sharing Attachments and Comments with Task Participants"
Limit allocated duration to (under the Advanced section)	Section 25.3.6.2.4, "Specifying a Time Limit for Acting on a Task"
Allow this participant to invite other participants (under the Advanced section)	Section 25.3.6.2.5, "Inviting Additional Participants to a Task"
Specify skip rule (under the Advanced section)	Section 25.3.6.2.6, "Bypassing a Task Participant"

25.3.6.2.1 Specifying the Voting Outcome

To specify group voting details:

1. Go to the **Vote Outcome** section of the Edit Participant Type dialog for the parallel type.
2. In the **Default Outcome** list, select the default outcome or enter an XPath expression for this task to take effect if the consensus percentage value is not satisfied. This happens if there is a tie or if all participants do not respond before the task expires. Seeded and custom outcomes that you entered in the Outcomes dialog in [Section 25.3.4.3, "Specifying a Task Outcome"](#) display in this list.
3. In the **Consensus Percentage** list, select a method for determining the outcome of the final task.
 - **By Number:** Select a percentage value or enter an XPath expression required for the outcome of this task to take effect (for example, a majority vote (**51**) or a unanimous vote (**100**)). For example, assume there are two possible outcomes (**ACCEPT** and **REJECT**) and five subtasks. If two subtasks are accepted and three are rejected, and the required acceptance percentage is 50%, the outcome of the task is rejected.

Note that this functionality is nondeterministic. For example, selecting a percentage of 30% when there are two subtasks does not make sense.
 - **By Expression:** Dynamically specify the details by clicking the icon to the right of the field to display the Expression Builder dialog.
4. Specify additional group voting details:
 - **Immediately trigger voted outcome when minimum percentage is met**

If selected, the outcome of the task can be computed early with the outcomes of the completed subtasks, enabling the pending subtasks to be withdrawn. For example, assume four users are assigned to act on a task, the default outcome is **APPROVE**, and the consensus percentage is set at **50**. If the first two users approve the task, the third and fourth users do not need to act on the task, since the consensus percentage value has been satisfied.

- **Wait until all votes are in before triggering outcome**

If selected, the workflow waits for all responses before an outcome is initiated.

25.3.6.2.2 Creating a Parallel Task Participant List

Users assigned to the list of participants can act upon tasks. You can create several types of lists:

- Value-based name and expression lists
- Value-based management chain lists
- Rule-based names and expression lists
- Rule-based management chain lists

For information about creating these lists of participants, see section [Section 25.3.6.1.1, "Creating a Single Task Participant List."](#)

25.3.6.2.3 Sharing Attachments and Comments with Task Participants

You can share comments and attachments with all group collaborators or workflow participants for a task. This information typically displays in the footer region of Oracle BPM Worklist.

1. Select **Share attachments and comments**.

25.3.6.2.4 Specifying a Time Limit for Acting on a Task

You can specify the amount of time a user, group, or application role receives to act on a task. If the user, group, or role does not act in the time specified, the global escalation and renewal policies that you set in the **Expiration and Escalation Policy** section (known as the routing slip level) of the Human Task Editor are applied. For example, if the global policy is set to escalate the task and this participant does not act in the duration provided, the task is escalated to the manager or another user, as appropriate.

To specify a time limit for acting on a task:

1. In the **Advanced** section of the Edit Participant Type dialog for the parallel type, click the **Open** icon to expand the section shown in [Figure 25-37](#) on page 25-31.
2. Select **Limit allocated duration to**.
3. Specify the amount of time.

For more information about setting the global escalation and renewal policies in the **Expiration and Escalation Policy** section of the Human Task Editor, see [Section 25.3.8, "How to Escalate, Renew, or End the Task."](#)

25.3.6.2.5 Inviting Additional Participants to a Task

You can allow a task assignee to invite other participants into the workflow before routing it to the next assignee in this workflow. For example, assume the approval workflow goes from James Cooper to John Steinbeck. If this option is checked, James Cooper can decide to first route it to Irving Stone before it goes to John Steinbeck.

To invite additional participants to a task:

1. In the **Advanced** section of the Edit Participant Type dialog for the parallel type, click the **Open** icon to expand the section (if not expanded).
2. Select the **Allow this participant to invite other participants**.

25.3.6.2.6 Bypassing a Task Participant

You can bypass a task participant (user, group, or application role) if a specific condition is satisfied. For example, if a user submits a business trip expense report that is under a specific amount, no approval is required by their manager.

To bypass a task participant:

1. In the Edit Participant Type dialog for the parallel type, select the **Specify skip rule** checkbox.

This action displays an icon for accessing the Expression Builder dialog for building a condition. The expression must evaluate to a boolean value.

For information about a valid XPath expression for skipping a participant, see [Section 25.3.6.1.4, "Bypassing a Task Participant."](#)

25.3.6.3 Configuring the Serial Participant Type

[Figure 25–38](#) displays the Serial dialog.

This participant type enables you to create a list of sequential participants for a workflow. For example, if you want a document to be reviewed by John, Mary, and Scott in sequence, use this participant type. For the serial participant type, they can be any list of users or groups.

Figure 25–38 Edit Participant Type — Serial Type

Type: Serial Label: Stage1.Participant1
e.g., Approval Manager

Participant List

Build a list of participants using: Names and expressions

Specify attributes using: Value-based Rule-based

Identification Type	Data Type	Value

Builds a serial list of the specified participants

Advanced

Limit allocated duration to:

Day Hour Minutes

Over all task duration is currently set to {0} days, {1} hours, and {2} minutes.

Allow this participant to invite other participants

Specify skip rule

To configure the serial participant type:

1. In the **Label** field, enter a recognizable label for this participant. This label must be unique among all the participants in the task definition (for example, *Approval Manager*, *Primary Reviewers*, and so on).

Instructions for configuring the following subsections of the Edit Participant Type dialog for the serial participant type are listed in [Table 25–8](#).

Table 25–8 Edit Participant Type — Serial Type

For This Subsection...	See...
Participant List	Section 25.3.6.3.1, "Creating a Serial Task Participant List"
Limit allocated duration to (under the Advanced section)	Section 25.3.6.3.2, "Specifying a Time Limit for Acting on a Task"
Allow this participant to invite other participants (under the Advanced section)	Section 25.3.6.3.3, "Inviting Additional Participants to a Task"
Specify skip rule (under the Advanced section)	Section 25.3.6.3.4, "Bypassing a Task Participant"

25.3.6.3.1 Creating a Serial Task Participant List

Users assigned to the list of participants can act upon tasks. You can create several types of lists:

- Value-based name and expression lists
- Value-based management chain lists
- Rule-based names and expression lists
- Rule-based management chain lists

See section [Section 25.3.6.1.1, "Creating a Single Task Participant List"](#) for instructions on creating these lists of participants.

25.3.6.3.2 Specifying a Time Limit for Acting on a Task

You can specify the amount of time a user, group, or application role receives to act on a task. If the user, group, or role does not act in the time specified, the global escalation and renewal policies that you set in the **Expiration and Escalation Policy** section (known as the routing slip level) of the Human Task Editor are applied. For example, if the global policy is set to escalate the task and this participant does not act in the duration provided, the task is escalated to the manager or another user, as appropriate.

To specify a time limit for acting on a task:

1. In the **Advanced** section of the Edit Participant Type dialog for the **Serial** type, click the **Open** icon to expand the section shown in [Figure 25–38](#).
2. Click **Limit allocated duration to**.
3. Specify the amount of time.

For more information about setting the global escalation and renewal policies in the **Expiration and Escalation Policy** section of the Human Task Editor, see [Section 25.3.8, "How to Escalate, Renew, or End the Task."](#)

25.3.6.3.3 Inviting Additional Participants to a Task

You can allow a task assignee to invite other participants into the workflow before routing it to the next assignee in this workflow. For example, assume the approval workflow goes from James Cooper to John Steinbeck. If this option is checked, James Cooper can decide to first route it to Irving Stone before it goes to John Steinbeck.

To invite additional participants to a task:

1. In the **Advanced** section of the Edit Participant Type dialog for the serial type, click the **Open** icon to expand the section (if not expanded).
2. Select **Allow this participant to invite other participants**.

Note: For the serial participant type, additional participants can be invited as follows:

- Globally specifying that the ad hoc participants can be invited at anytime. In this case, even in a sequential workflow, approvers can invite other participants at any level in the sequential workflow.
 - Specifying that an ad hoc invitation of other participants can be done only in specific points in the workflow. In this case, other ad hoc participants are invited only when a serial in complete.
-
-

25.3.6.3.4 Bypassing a Task Participant

You can bypass a task participant (user, group, or application role) if a specific condition is satisfied. For example, if a user submits a business trip expense report that is under a specific amount, no approval is required by their manager.

To bypass a task participant:

1. In the **Advanced** section of the Edit Participant Type dialog for the serial type, select the **Specify skip rule** checkbox.

This action displays an icon for accessing the Expression Builder dialog for building a condition. The expression must evaluate to a boolean value.

For more information about a valid XPath expression for skipping a participant, see [Section 25.3.6.1.4, "Bypassing a Task Participant."](#)

25.3.6.4 Configuring the FYI Participant Type

[Figure 25–39](#) displays the Edit Participant Type dialog for the FYI type.

This participant type is used when a task is sent to a user, but the business process does not wait for a user response; it just continues. FYIs cannot directly impact the outcome of a task, but in some cases can provide comments or add attachments.

For example, a magazine subscription is due for renewal. If the user does not cancel the current subscription before the expiration date, the subscription is renewed. This user is reminded weekly until the request expires or the user acts on it.

Figure 25–39 Edit Participant Type — FYI Type

Type: FYI Label: Stage1.Participant1
e.g., Approval Manager

Participant List

Build a list of participants using: Names and expressions

Specify attributes using: Value-based Rule-based

Participant Names		
Identification Type	Data Type	Value

To configure the FYI participant type:

1. In the **Label** field, enter a recognizable label for this participant. This label must be unique among all the participants in the task definition (for example, *Approval Manager*, *Primary Reviewers*, and so on).

25.3.6.4.1 Creating an FYI Task Participant List

Users assigned to the list of participants can act upon tasks. You can create several types of lists:

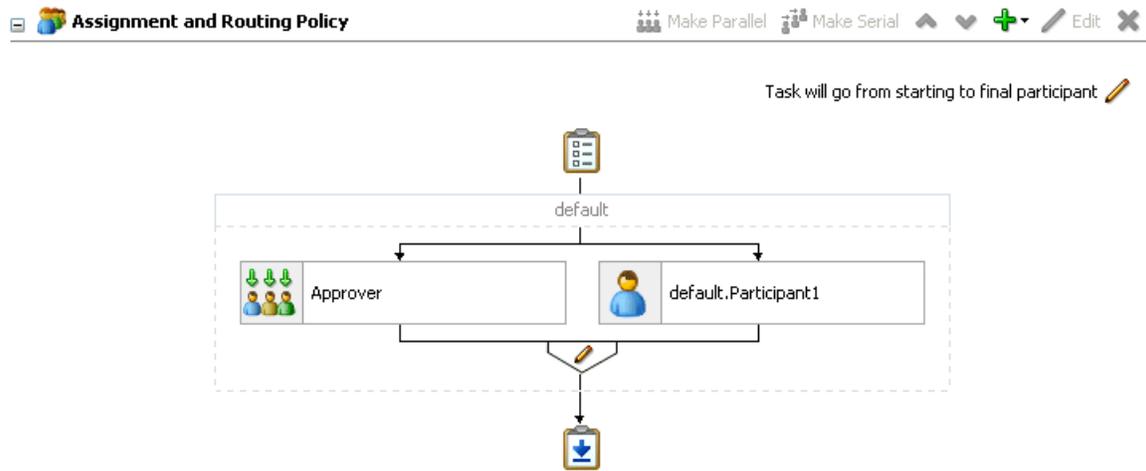
- Value-based name and expression lists
- Value-based management chain lists
- Rule-based names and expression lists
- Rule-based management chain lists

See section [Section 25.3.6.1.1, "Creating a Single Task Participant List"](#) for instructions on creating these lists of participants.

25.3.7 How to Select a Routing Policy

After you configure a participant type and are returned to the Human Task Editor, the **Task will go from starting to final participant** icon is enabled, as shown in [Figure 25–40](#).

Figure 25–40 Human Task Editor — Assignment and Routing Policy Section



Click this icon to display the Configure Assignment dialog shown in [Figure 25–41](#). This dialog enables you to specify a method for routing your task through the workflow.

Figure 25–41 Configure Assignment

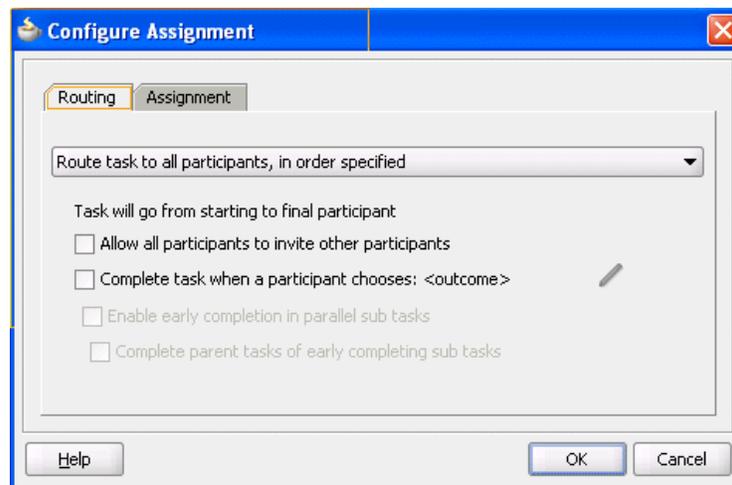


Table 25–9 describes the routing policy methods provided.

Table 25–9 Routing Policy Method

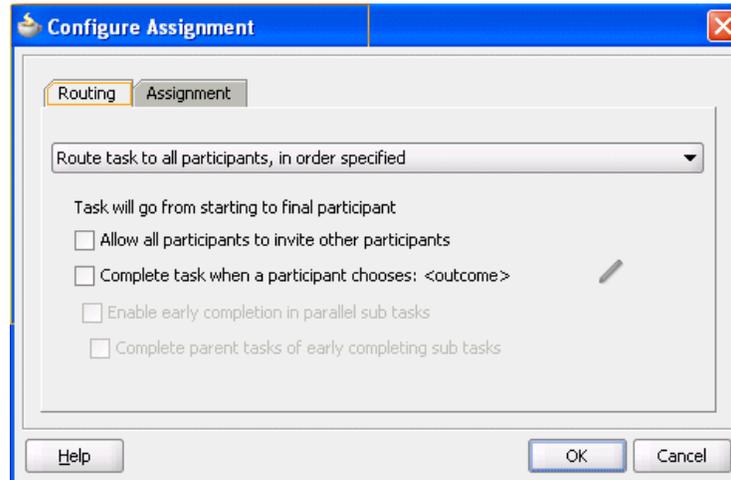
Routing Policy Selection	Use This Policy In Environments Where...	Section
<p>Route task to all participants, in order specified</p> <p>This selection enables you to specify the following suboptions:</p>	<p>A task must be routed to each of the participants in the order in which they appear. This is predetermined, default routing. For example, in a hiring process, if three users interview and provide review feedback, then the task is sent to the human resources department.</p>	<p>Section 25.3.7.1, "Routing Tasks to All Participants in the Specified Order"</p>
<ul style="list-style-type: none"> ▪ Allow all participants to invite other participants 	<p>A participant can select users or groups as the next assignee (ad hoc) when approving the task.</p>	<p>Section 25.3.7.1.1, "Allowing All Participants to Invite Other Participants"</p>
<ul style="list-style-type: none"> ▪ Complete task when a participant chooses <outcome> 	<p>A participant in a task can accept or reject it, thus ending the workflow without the task being sent to any other participant. For example, a manager rejects a purchase order, meaning that purchase order is not sent to their manager for review.</p>	<p>Section 25.3.7.1.2, "Stopping Routing of a Task to Further Participants"</p>
<ul style="list-style-type: none"> ▪ Enabling Early Completion in Parallel Subtasks 	<p>Note: This option is for environments in which you have multiple stages and participants working in parallel.</p> <p>Participants perform subtasks in parallel, and one group's rejection or approval of a subtask does not cause the other group's subtask to also be rejected or approved.</p>	<p>Section 25.3.7.1.3, "Enabling Early Completion in Parallel Subtasks"</p>
<ul style="list-style-type: none"> ▪ Completing Parent Subtasks of Early Completing Subtasks 	<p>Note: This option is for environments in which you have multiple stages and participants working in parallel.</p> <p>Participants perform subtasks in parallel, and one group's rejection or approval of a subtask causes the other group's subtask to also be rejected or approved.</p>	<p>Section 25.3.7.1.4, "Completing Parent Subtasks of Early Completing Subtasks"</p>
Use Advanced Rules	<p>The participants to whom the task is routed are determined by the business rule logic that you model. For example, a loan application task is designed to go through a loan agent, their manager, and then the senior manager. If the loan agent approves the loan, but their manager rejects it, the task is returned to the loan agent.</p>	<p>Section 25.3.7.2, "Specifying Advanced Task Routing Using Business Rules"</p>
Use External Routing	<p>The participants in a task are dynamically determined. For example, a company's rules may require the task participants to be determined and then retrieved from a back-end database during runtime.</p>	<p>Section 25.3.7.3, "Using External Routing"</p>
Assignment tab	<p>A participant is assigned a failed task for the purposes of recovery.</p>	<p>Section 25.3.7.4, "Configuring the Error Assignee"</p>

25.3.7.1 Routing Tasks to All Participants in the Specified Order

You can select to have a task reviewed by all selected participants. This is known as default routing because the task is routed to each of the participants in the order in which they appear. This type of routing differs from state machine-based routing.

To route tasks to all participants in the specified order:

1. In the **Assignment and Routing Policy** section, click the icon to the right of **Task will go from starting to final participant**.
2. Select **Route task to all participants, in order specified** from the list shown in [Figure 25-42](#).

Figure 25-42 Route a Task to All Participants

See the following sections for instructions on defining a routing policy:

- Allowing all participants to invite other participants
- Completing a task when a participant chooses
- Enabling early completion in parallel subtasks
- Completing parent subtasks of early completing subtasks

25.3.7.1.1 Allowing All Participants to Invite Other Participants This checkbox is the equivalent of the ad hoc workflow pattern of pre-10.1.3 Oracle BPEL Process Manager releases. This applies when there is at least one participant. In this case, each user selects users or groups as the next assignee when approving the task.

To allow all participants to invite other participants:

1. In the **Assignment and Routing Policy** section, click the icon to the right of **Task will go from starting to final participant**.
2. Select **Route task to all participants, in order specified**.
3. Select the **Allow all participants to invite other participants** checkbox for this task assignee to invite other participants into the workflow before routing it to the next assignee in this workflow.

25.3.7.1.2 Stopping Routing of a Task to Further Participants You can specify conditions under which to complete a task early, regardless of the other participants in the workflow.

For example, assume an expense report goes to the manager, and then the director. If the first participant (manager) rejects it, you can end the workflow without sending it to the next participant (director).

To abruptly complete a condition:

1. In the **Assignment and Routing Policy** section, click the icon to the right of **Task will go from starting to final participant**.
2. Select **Route task to all participants, in order specified** from the list.
3. Select the **Complete task when a participant chooses <outcome>** checkbox.

The Abrupt Completion Details dialog appears.

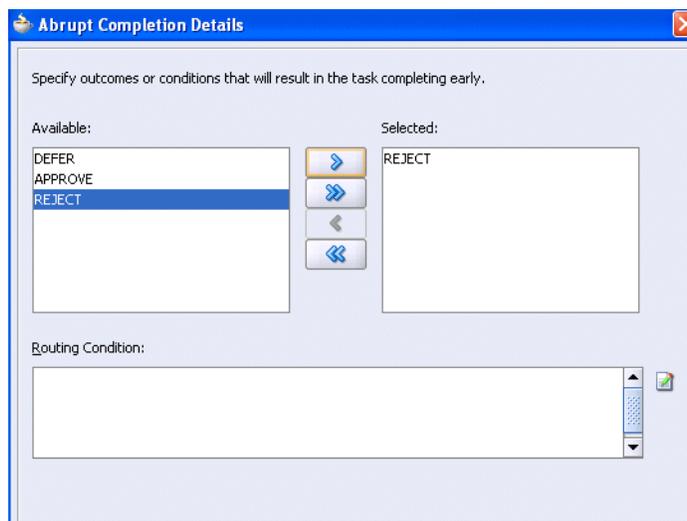
There are two methods for specifying the abrupt completion of a task:

- Outcomes
- XPath expression routing condition

If outcomes are specified, any time the selected task outcome occurs, the task completes. If both outcome and routing condition are specified, the workflow service performs a logical OR on the two.

4. Select appropriate outcomes and click the > button, as shown in [Figure 25–43](#). To select all, click the >> button.

Figure 25–43 Abrupt Completion Details



5. To the right of the **Routing Condition** field, click the icon to display the Expression Builder dialog for dynamically creating a condition under which to complete this task early. For example, if a user submits a business trip expense report that is under a specific amount, no approval is required by their manager.
6. If you want to enable early completion, click **Enable early completion in parallel with subtasks**. For more information, see [Section 25.3.7.1.3, "Enabling Early Completion in Parallel Subtasks."](#)
7. If you want to enable early completion of parent tasks, click **Complete parent tasks of early completing subtasks**. For more information, see [Section 25.3.7.1.4, "Completing Parent Subtasks of Early Completing Subtasks."](#)
8. Click **OK** to return to the Human Task Editor.

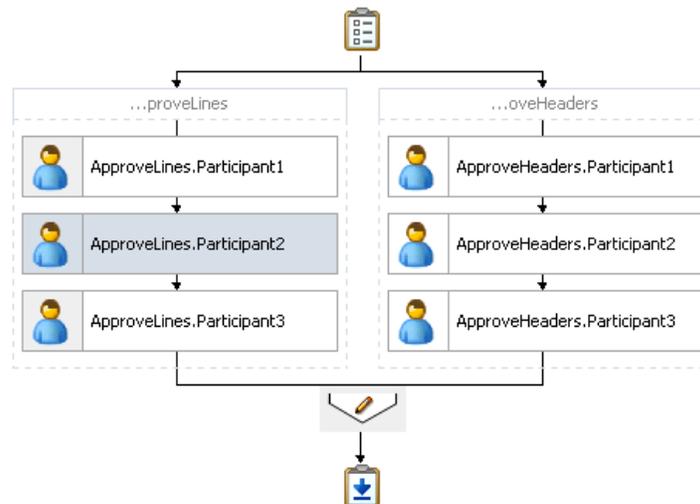
You can click the icon to the right of the **Complete task when a participant chooses <outcome>** checkbox to edit this information.

25.3.7.1.3 Enabling Early Completion in Parallel Subtasks You can use this option in the following environments:

- Multiple stages and groups of participants perform subtasks in parallel.
- A participant in one group approves or rejects a subtask, which causes the other participants in that same group to stop acting upon the task. However, this does not cause the other parallel group to stop acting upon subtasks. That group continues taking actions on tasks.

For example, assume there are two parallel subgroups, each in separate stages. One group acts upon lines of a purchase order. The other group acts upon headers of the same purchase order. If participant **ApproveLines.Participant2** of the first group rejects a line, all other task participants in the first group stop acting upon tasks. However, the second parallel group continues to act upon headers in the purchase order. In this scenario, the entire task does not complete early. [Figure 25–44](#) provides details.

Figure 25–44 Early Completion of Parallel Subtasks



25.3.7.1.4 Completing Parent Subtasks of Early Completing Subtasks You can use this option in the following environments:

- Multiple stages and groups of participants perform subtasks in parallel.
- A participant in one group approves or rejects a subtask, which causes the other participants in that same group to stop acting upon the task. This also causes the other parallel group to stop acting upon subtasks.

For example, assume there are two parallel subgroups, each in separate stages, as shown in [Figure 25–44](#). One group acts upon lines of a purchase order. The other group acts upon headers of the same purchase order. If participant **ApproveLines.Participant2** of the first group rejects a line, all other task participants in the first group stop acting upon tasks. In addition, the second parallel group stops acting upon headers in the purchase order. In this scenario, the entire task completes early.

25.3.7.2 Specifying Advanced Task Routing Using Business Rules

Use advanced routing rules to create complex workflow routing scenarios. The participant types (single, parallel, serial, and FYI) are used to create a linear flow from

one set of users to another with basic conditions such as abrupt termination, skipping assignees, and so on. However, there is often a need to perform more complex back and forth routing between multiple individuals in a workflow. One option is to use the BPEL process as the orchestrator of these tasks. Another option is to specify it declaratively using business rules. This section describes how you can model such complex interactions by using business rules with the Human Task Editor.

25.3.7.2.1 Introduction to Advanced Task Routing Using Business Rules You can define state machine routing rules using Oracle Business Rules. This action enables you to create Oracle Business Rules that are evaluated:

- After a routing slip task participant sets the outcome of the task
- Before the task is assigned to the next routing slip participant

This action enables you to override the standard task routing slip method described in [Section 25.3.7.1, "Routing Tasks to All Participants in the Specified Order"](#) and build complex routing behavior into tasks.

Using Oracle Business Rules, you define a set of rules (called a ruleset) that rely on business objects, called facts, to determine which action to take.

25.3.7.2.2 Facts A fact is an object with certain business data. Each time a routing slip assignee sets the outcome of a task, instead of automatically routing the task to the next assignee, the task service performs the following steps:

- Asserts facts into the decision service
- Executes the advanced routing rule set

Rules can test values in the asserted facts and specify the routing behavior by setting values in a `TaskAction` fact type.

[Table 25–10](#) describes the fact types asserted by the task service.

Table 25–10 Fact Types Asserted By the Task Service

Fact Type	Description
Task	This fact contains the current state of the workflow task instance. All task attributes can be tested against it. The task fact also contains the current task payload. This fact enables you to construct tests against payload values and task attribute values.
PreviousOutcome	This fact describes the previous task outcome and the assignee who set the outcome. The previous outcome fact contains the following attributes: <ul style="list-style-type: none"> ■ <code>actualParticipant</code>: The name of the participant who set the task outcome (for example, <code>jstein</code>) ■ <code>logicalParticipant</code>: The logical name (or label) for the routing slip participant responsible for setting the task outcome (for example, <code>assignee1</code>) ■ <code>outcome</code>: The outcome that was set (for example, <code>approve</code> or <code>reject</code>) ■ <code>level</code>: If the previous participant was part of a management chain, then this attribute records their level in the chain, where 1 is the first level in the chain. For other participant types, the value is -1. ■ <code>totalNumberOfApprovals</code>: The total number of users that have now set the outcome of the task.

Table 25–10 (Cont.) Fact Types Asserted By the Task Service

Fact Type	Description
TaskAction	This fact is not intended for writing rule tests against it. Instead, it is updated by the ruleset, and returned to the task service to indicate how the task should be routed. Rules should not directly update the TaskAction fact. Instead, they should call one of the RL functions described in Section 25.3.7.2.3, "Action Types." These functions handle updating the TaskAction fact with the appropriate values.

Some fact types can only be used in workflow routing rules, while others can only be used in workflow participant rules. [Table 25–11](#) describes where you can use each type.

Table 25–11 Use of Fact Types

Fact Type	Can Use in Routing Rules?	Can Use in Participant Rules?
Task	Yes	Yes
PreviousOutcome	Yes	No
TaskAction	Yes	No
Lists	No	Yes
RoutingSlipObjectFactory	No	Yes
ResourceListType	No	Yes
ManagementChainListType	No	Yes
ResourceType	No	Yes
ParameterType	No	Yes
AutoActionType	No	Yes
ResponseType	No	Yes

25.3.7.2.3 Action Types To instruct the task service on how to route the task, rules can specify one of many task actions. This is done by updating the TaskAction fact asserted into the rule session. However, rules should not directly update the TaskAction fact. Instead, rules should call one of the action RL functions, passing the TaskAction fact as a parameter. These functions handle the actual updates to the fact. For example, to specify an action of go forward, you must add a `call GO_FORWARD(TaskAction)` to the action part of the rule.

Each time a state machine routing rule is evaluated, the rule takes one of the actions shown in [Table 25–12](#):

Table 25–12 Business Rule Actions

Action	Description	Parameters
GO_FORWARD	Goes to the next participant in the routing slip (default behavior).	None
PUSHBACK	Goes back to the previous participant in the routing slip (the participant before the one that just set the task outcome).	None

Table 25–12 (Cont.) Business Rule Actions

Action	Description	Parameters
GOTO	Goes to a specific participant in the routing slip.	participant' A string that identifies the label of the participant (for example, Approver1) to which to route the task.
COMPLETE	Finishes routing and completes the task. The task is marked as completed, and no further routing is required.	None
ESCALATE	Escalates and reassigns the task according to the task escalation policy (usually to the manager of the current assignee).	None

25.3.7.2.4 Sample Rule Set This section describes how to use rules to implement custom routing behavior with a simple example. A human workflow task is created for managing approvals of expense requests. The outcomes for the task are approve and reject. The task definition includes an `ExpenseRequest` payload element. One of the fields of `ExpenseRequest` is the total amount of the expense request. The routing slip for the task consists of three single participants (`assignee1`, `assignee2`, and `assignee3`).

By default, the task gets routed to each of the assignees, with each assignee choosing to approve or reject the task.

Instead of this behavior, the necessary routing behavior is as follows:

- If the total amount of the expense request is less than \$100, approval is only required from one of the participants. Otherwise, it must be approved by all three.
- If an expense request is rejected by any of the participants, it must be returned to the previous participant for re-evaluation. If it is rejected by the first participant, the expense request is rejected and marked as completed.

This behavior is implemented using the following rules. Note that when a rule dictionary is generated for advanced routing rules, it is created with a template rule that implements the default `GO_FORWARD` behavior. You can edit this rule, and make copies of the template rule by right-clicking and selecting **Copy Rule** in the Oracle Business Rules Designer.

If the amount is greater than \$100 and the previous assignee approved the task, it is not necessary to provide a rule for routing a task to each of the assignees in turn. This is the default behavior that is reverted to if none of the rules in the rule set are triggered:

- Early approval rule (Figure 25–45):

Figure 25–45 Early Approval Rule

```

Rule_1
  Allow early approval of low-cost expense requests

IF
  Task.payload.expenseRequest.amount < 100 and
  PreviousOutcome.outcome == "APPROVE"
  <insert test>

THEN
  call COMPLETE()
  <insert action>

```

- Push back on the rejected rule (Figure 25–46):

Figure 25–46 Push Back On The Rejected Rule

```

Rule_2
  Pushback to previous assignee on rejection of request

IF
  PreviousOutcome.outcome == "REJECT" and
  PreviousOutcome.logicalParticipant != "Assignee1"
  <insert test>

THEN
  call PUSHBACK()
  <insert action>

```

- Complete the Assignee1 rejected rule (Figure 25–47):

Figure 25–47 Completion of the Assignee1 Rejected Rule

```

Rule_3
  Complete task if first assignee rejects request

IF
  PreviousOutcome.outcome == "REJECT" and
  PreviousOutcome.logicalParticipant == "Assignee1"
  <insert test>

THEN
  call COMPLETE()
  <insert action>

```

For information about iterative design, see the workflow-106-IterativeDesign sample available at the Oracle Technology Network:

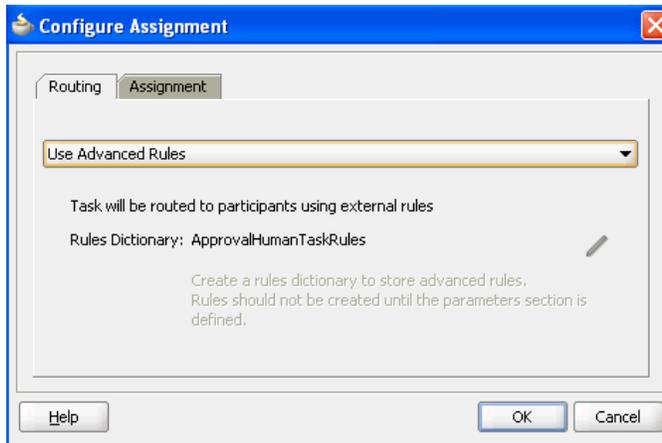
http://www.oracle.com/technology/sample_code/products/hwf

25.3.7.2.5 Creating Advanced Routing Rules

To create advanced routing rules:

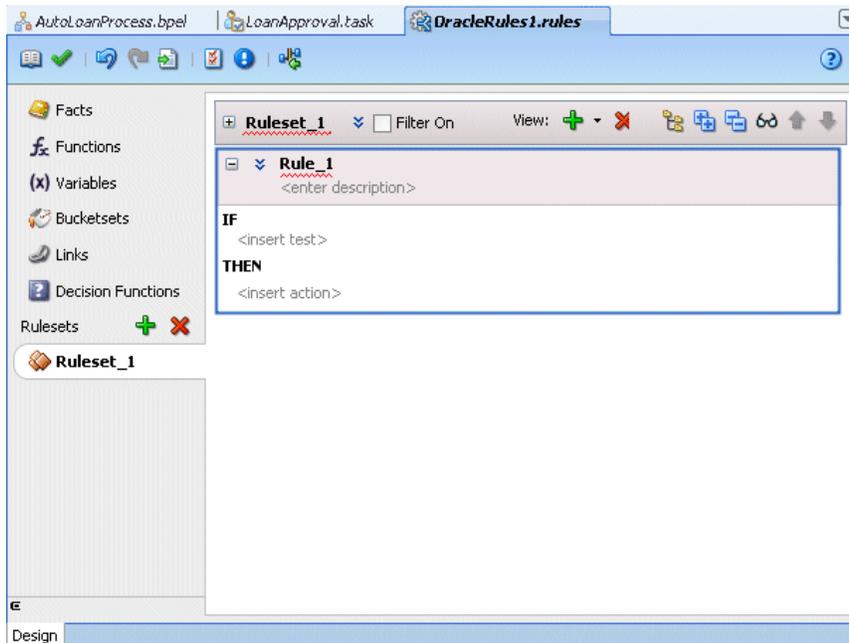
1. In the **Assignment and Routing Policy**, click the icon to the right of **Task will go from starting to final participant**.
2. Select **Use Advanced Rules** from the list.
3. To the right of **Rules Dictionary**, click the **Edit** icon, as shown in Figure 25–48.

Figure 25–48 Creating a Rules Dictionary



This starts the Oracle Business Rules Designer with a preseeded repository containing all necessary fact definitions, as shown in [Figure 25–49](#). A decision service component is created for the dictionary, and is associated with the task service component.

Figure 25–49 Human Task Rule Dictionary



4. Define state machine routing rules for your task using Oracle Business Rules.

This automatically creates a fully-wired decision service in the human task and the associated rule repository and data model.

For more information on business rules:

- An example human task ruleset, see [Section 25.3.7.2.4, "Sample Rule Set"](#)
- *Oracle Fusion Middleware User's Guide for Oracle Business Rules*
- *Oracle Fusion Middleware Language Reference Guide for Oracle Business Rules*

25.3.7.3 Using External Routing

You configure an external routing service that dynamically determines the participants in the workflow. If this routing policy is specified, all other participant types are ignored. It is assumed that the external routing service provides a list of participant types (single approver, serial approver, parallel approver, and so on) at runtime to determine the routing of the task.

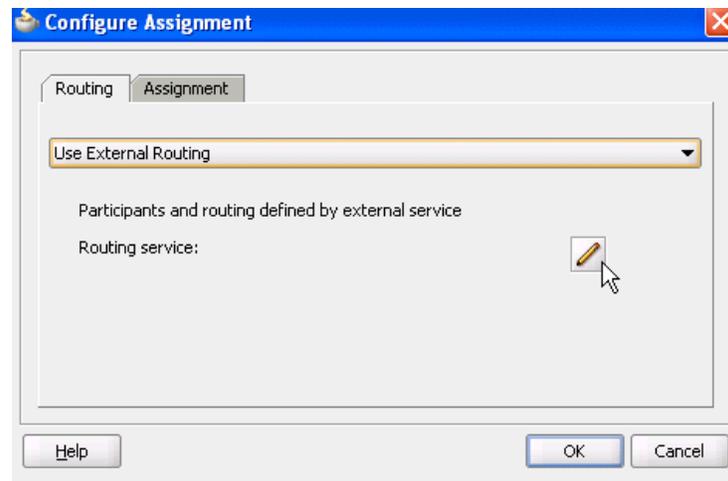
Use this option if you do not want to use any of the routing rules to determine task assignees. In this case, all the logic of task assignment is delegated to the external routing service.

Note: If you select **Use External Routing** in the Configure Assignment dialog, specify a Java class, and click **OK** to exit, the next time you open this dialog, the other two selections (**Route task to all participants, in order specified** and **Use Advanced Rules**) no longer appear in the dropdown list. To access all three selections again, you must delete the entire assignment.

To use external routing

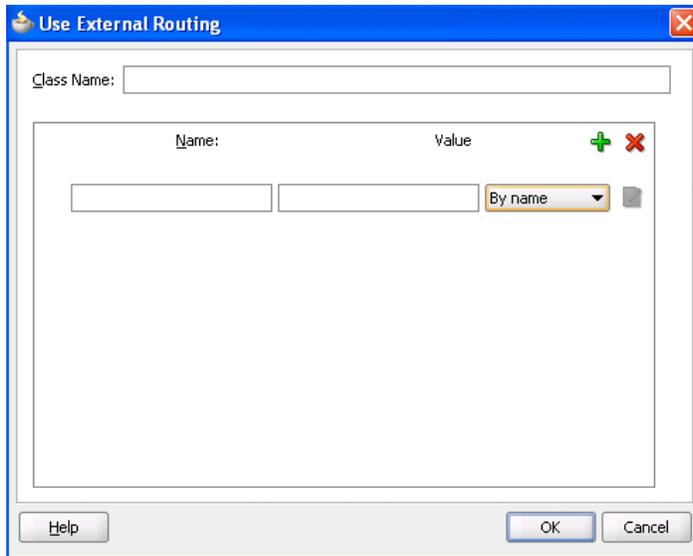
1. In the **Assignment and Routing Policy**, click the icon to the right of **Task will go from starting to final participant**.
2. Select **Use External Routing** from the list.
3. Click the **Edit** icon, as shown in [Figure 25-50](#).

Figure 25-50 Selection of Use External Routing



The External Routing dialog appears, as shown in [Figure 25-51](#).

Figure 25–51 Use External Routing Dialog



4. In the **Class Name** field, enter the fully qualified class file name (for example, the `org.mycompany.tasks.RoutingService` class name). This class must implement the `oracle.bpel.services.workflow.task.IAssignmentService` interface.
5. Add name and pair value parameters by name or XPath expression that can be passed to the external service, as shown in [Table 25–13](#).

Table 25–13 External Routing

Field	Description
By Name	Enter a name in the Name field and a value in the Value field.
By Expression	Enter a name and dynamically enter a value by clicking the icon to the right of the field to display the Expression Builder dialog.

6. Click the **Add** icon to add additional name and pair value parameters.

25.3.7.4 Configuring the Error Assignee

Tasks can error for reasons such as incorrect assignments. When such errors occur, the task is assigned to the error assignee, who can perform corrective actions. Recoverable errors are as follows:

- Invalid user and group for all participants
- Invalid XPath expressions that are related to assignees and expiration duration
- Escalation on expiration errors
- Evaluating escalation policy
- Evaluating renewal policy
- Computing management chain
- Evaluating dynamic assignment rules. The task is not currently in error, but is still left as assigned to the current user and is therefore recoverable.

- Dynamic assignment cyclic assignment (for example, user A > user B > user A). The task is not currently in error, but is still left as assigned to the last user in the chain and is therefore recoverable.

The following errors are not recoverable. In these cases, the task is moved to the terminating state `ERRORED`.

- Invalid task metadata
- Unable to read task metadata
- Invalid `GOTO` participant from state machine rules
- Assignment service not found
- Any errors from assignment service
- Evaluating custom escalate functions
- Invalid XPath and values for parallel default outcome and percentage values

During modeling of workflow tasks, you can specify error assignees for the workflow. If error assignees are specified, they are evaluated and the task is assigned to them. If no error assignee is specified at runtime, an administration user is discovered and is assigned the alerted task. The error assignee can perform one of the following actions:

- Ad hoc route
Route the task to the actual users assigned to the task. Ad hoc routing allows the task to be routed to users in sequence, parallel, and so on.
- Reassign
Reassign the task to the actual users assigned to this task
- Error task
Indicate that this task cannot be rectified.

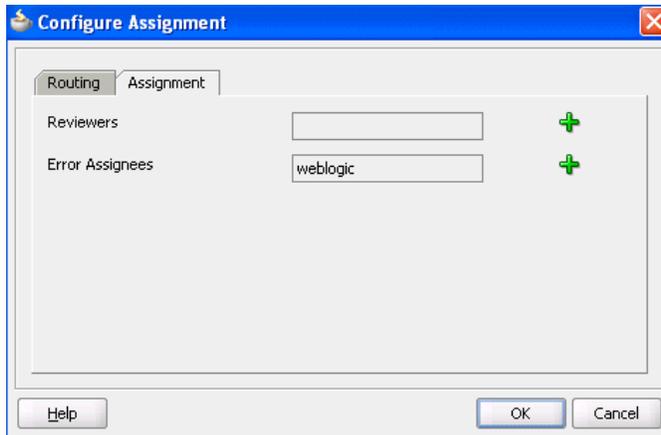
If there are any errors in evaluating the error assignees, the task is marked as being in error.

This dialog enables you to specify the users or groups to whom the task is assigned if an error in assignment has occurred.

To configure the error assignee:

1. In the **Assignment and Routing Policy** section, click the icon to the right of **Task will go from starting to final participant**.
2. Click the **Assignment** tab.
3. Click the **Add** icon to assign reviewers or error assignees, as shown in [Figure 25-52](#).

Figure 25–52 Error Assignment Details



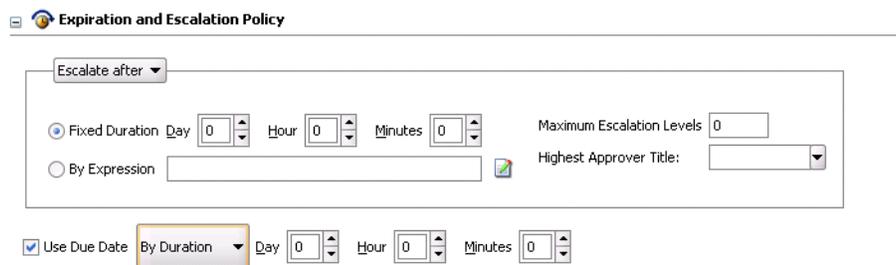
4. Click the **Add** icon and select a user, group, or application role to participate in this task.
 The **Identification Type** column of the **Starting Participant** table displays your selection of user, group, or application role.
5. See Step 4 through 6 of [Section 25.3.6.1.1, "Creating a Single Task Participant List"](#) for instructions on selecting a user, group, or application role.
 For more information about users, groups, or application roles, see [Section 24.2.1.1.3, "Participant Assignment."](#)

25.3.8 How to Escalate, Renew, or End the Task

[Figure 25–53](#) shows the **Expiration and Escalation Policy** section of the Human Task Editor.

You can specify expiration duration of a task in this global policy section (also known as the routing slip level). If expiration duration is specified at the routing slip level instead of at the participant type level, then this duration is the expiration duration of the task across all the participants. However, if you specify expiration duration at the participant type level (through the **Limit allocated duration to** field), then those settings take precedence over settings specified in the **Expiration and Escalation Policy** section (routing slip level).

Figure 25–53 Human Task Editor — Expiration and Escalation Policy Section



25.3.8.1 Introduction to Escalation and Expiration Policy

This section provides an overview of how specifying the expiration duration at this level makes this setting the expiration duration of the task across all the participants.

For example, participant **LoanAgentGroup** and participant **Supervisor** have three days to act on the task between them, as shown in [Figure 25–54](#):

Figure 25–54 *Expire After Policy*

If there is no expiration specified at either the participant level or this routing slip level, then that task has no expiration duration.

If expiration duration is specified at any of the participant's level, then for that participant, the participant expiration duration is used. However, the global expiration duration is still used for the participants that do not have participant level expiration duration. The global expiration duration is always decremented by the time elapsed in the task.

The policy for interpreting the participant level expiration for the participants is described as follows:

- **Serial**

Each assignment in the management chain gets the same expiration duration as the one specified in the serial. Note that the duration is not for all the assignments resulting from this assignment. If the task expires at any of the assignments in the management chain, the escalation and renewal policy is applied.

- **Parallel:**

- In a parallel workflow, if the parallel participants are specified as a resource, a routing slip is created for each of the resources. The expiration duration of each created routing slip follows these rules:
 - * The expiration duration equals the expiration duration of the parallel participant if it has an expiration duration specified.
 - * The expiration duration that is left on the task if it was specified at the routing slip level.
 - * Otherwise, there is no expiration duration.
- If parallel participants are specified as routing slips, then the expiration duration for the parallel participants are determined by the routing slip.

Note: When the parent task expires in a parallel task, the subtasks are withdrawn if those tasks have not expired or completed.

25.3.8.2 Specifying a Policy to Never Expire

You can specify for a task to never expire.

To specify a policy to never expire:

1. In the dropdown list in the **Expiration and Escalation Policy** section, select **Never Expire**, as shown in [Figure 25-53](#).

25.3.8.3 Specifying a Policy to Expire

You can specify for a task to expire. When the task expires, either the escalation policy or the renewal policy at the routing slip level is applied. If neither is specified, the task expires. The expiration policy at the routing slip level is common to all the participants.

To specify for a task to expire:

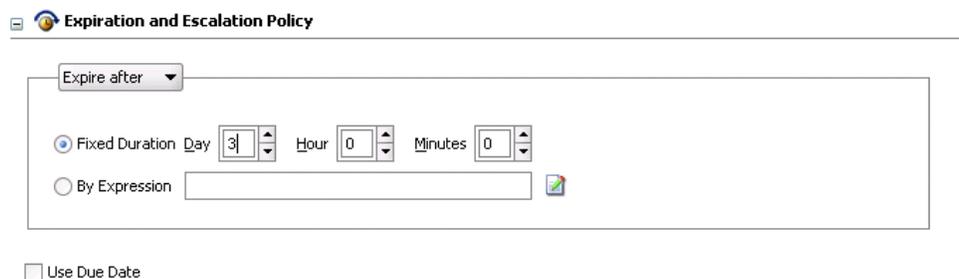
1. In the dropdown list in the **Expiration and Escalation Policy** section, select **Expire after**, as shown in [Figure 25-53](#).
2. Specify the maximum time period for the task to remain open.

The expiration policy for parallel participants is interpreted as follows:

- If parallel participants are specified as resources in parallel elements, there is no expiration policy for each of those participants.
- If parallel participants are specified as routing slips, then the expiration policy for the routing slip applies to the parallel participants.

[Figure 25-55](#) indicates that the task expires in three days.

Figure 25-55 Expire After Policy



25.3.8.4 Extending an Expiration Policy Period

You can extend the expiration period when the user does not respond within the allotted time. You do this by specifying the number of times the task can be renewed upon expiration (for example, renew it an additional three times) and the duration of each renewal (for example, three days for each renewal period).

To extend an expiration policy period:

1. In the dropdown list in the **Expiration and Escalation Policy** section, select **Renew after**, as shown in [Figure 25-53](#) on page 25-52.
2. Specify the maximum number of times to continue renewing this task.

In [Figure 25-56](#), when the task expires, it is renewed at most three times. It does not matter if the task expired at the **LoanAgentGroup** participant or the **Supervisor** participant.

Figure 25–56 Renew After Policy

The screenshot shows the 'Expiration and Escalation Policy' configuration window. The 'Renew after' dropdown is selected. Below it, the 'Fixed Duration' radio button is selected, with 'Day' set to 3, 'Hour' to 0, and 'Minutes' to 0. The 'Maximum Renewals' field is set to 3. The 'By Expression' radio button is unselected, and its corresponding text box is empty.

25.3.8.5 Escalating a Task Policy

You can escalate a task if a user does not respond within the allotted time. For example, if you are using the escalation hierarchy configured in your user directory, the task can be escalated to the user’s manager. If you are using escalation callbacks, the task is escalated to whoever you have defined. When a task has been escalated the maximum number of times, it stops escalating. An escalated task can remain in a user inbox even after the task has expired.

To escalate a task policy:

1. In the dropdown list in the **Expiration and Escalation Policy** section, select **Escalate after**, as shown in [Figure 25–53](#) on page 25-52.
2. Specify the following additional values. When both are set, the escalation policy is more restrictive.

- #### ■ Maximum Escalation Levels

Number of management levels to which to escalate the task. This field is required.

- #### ■ Highest Approver Title

The title of the highest approver (for example, self, manager, director, or CEO). These titles are compared against the title of the task assignee in the corresponding user repository. This field is optional.

The escalation policy specifies the number of times the task can be escalated on expiration and the renewal duration. In [Figure 25–57](#), when the task expires, it is escalated at most three times. It does not matter if the task expired at the **LoanAgentGroup** participant or the **Supervisor** participant.

Figure 25–57 Escalate After Policy

The screenshot shows the 'Expiration and Escalation Policy' configuration window. The 'Escalate after' dropdown is selected. Below it, the 'Fixed Duration' radio button is selected, with 'Day' set to 3, 'Hour' to 0, and 'Minutes' to 0. The 'Maximum Escalation Levels' field is set to 0. The 'Highest Approver Title' dropdown is set to 'Director'. The 'By Expression' radio button is unselected, and its corresponding text box is empty.

25.3.8.6 Specifying a Due Date

A due date is used to indicate the date by which the task should be completed. Note that the due date is different from the expiration date. When a task expires it is either marked expired or automatically escalated or renewed based on the escalation policy.

The due date is generally a date earlier than the expiration date and an indication to the user that the task is about to expire.

You can enter a due date for a task, as shown in [Figure 25–53](#). A task is considered overdue after it is past the specified due date. This date is in addition to the expiration policy. A due date can be specified irrespective of whether an expiration policy has been specified. The due date enables Oracle BPM Worklist to display a due date, list overdue tasks, highlight overdue tasks in the inbox, and so on. Overdue tasks can be queried using a predicate on the `TaskQueryService.queryTask(...)` API.

To specify a due date:

1. In the **Expiration and Escalation Policy** section, select the **Use Due Date** checkbox.
2. Select **By Date** to enter a specific due date or select **By Expression** to dynamically enter a value as an XPath expression.

Note the following details:

- The due date can be set on both the task (using the Create ToDo Task dialog in Oracle BPM Worklist) and in the `.task` file (using the Human Task Editor). This is to allow to-do tasks without task definitions to set a due date during initiation of the task. A due date that is set in the task (a runtime object) overrides a due date that is set in the `.task` file.
- In the task definition, the due date can only be specified at the global level, and not for each participant.
- If the due date is set on the task, the due date in the `.task` file is ignored.
- If the due date is not set on the task, the due date in the `.task` file is evaluated and set on the task.
- If there is no due date on either the task or in the `.task` file, there is no due date on the task.

Note: You cannot specify business rules for to-do tasks.

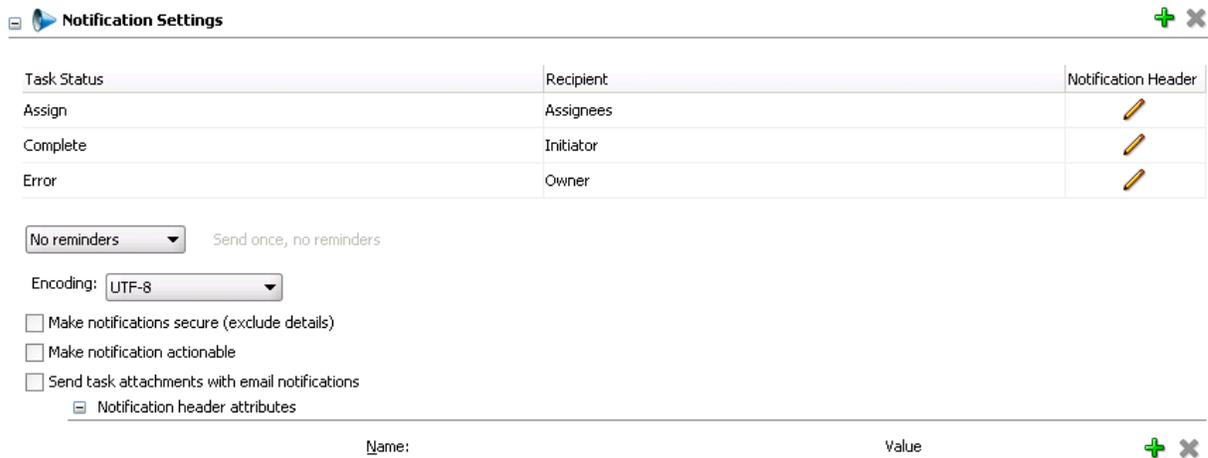
For more information, see [Section 27.3.4, "How To Create a ToDo Task."](#)

25.3.9 How to Specify Participant Notification Preferences

[Figure 25–58](#) shows the **Notification Settings** section of the Human Task Editor (when fully expanded).

Notifications indicate when a user is assigned a task or informed that the status of the task has changed. Notifications can be sent through email, voice message, instant message, or SMS. Notifications are sent to different types of participants for different actions. Notifications are configured by default with default messages. For example, a notification message is sent to indicate that a task has completed and closed. You can create your own or modify existing configurations.

Figure 25–58 Human Task Editor — Notification Settings Section



To specify participant notification preferences:

1. In the **Notification Settings** section, click the **Open** icon to expand the section (displays as shown in [Figure 25–58](#)).

Instructions for configuring the following subsections of the **Notification Settings** section are listed in [Table 25–14](#).

Table 25–14 Human Task Editor — Notification Settings Section

For This Subsection...	See...
Task Status	Section 25.3.9.1, "Notifying Recipients of Changes to Task Status"
Recipient	Section 25.3.9.2, "Editing the Notification Message"
Reminders	Section 25.3.9.3, "Setting Up Reminders"
Encoding	Section 25.3.9.4, "Changing the Character Set Encoding"
Make notifications secure (exclude details)	Section 25.3.9.5, "Securing Notifications to Exclude Details"
Make notifications actionable	Section 25.3.9.5, "Securing Notifications to Exclude Details"
Send task attachments with email notifications	Section 25.3.9.6, "Making Email Messages Actionable"

For information about the notification service, see [Section 29.2, "Notifications from Human Workflow."](#)

25.3.9.1 Notifying Recipients of Changes to Task Status

Three default status types display in the **Task Status** column: **Assign**, **Complete**, and **Error**. You can select other status types for which to receive notification messages.

To notify recipients of changes to task status:

1. In the **Task Status** column of the **Notification Settings** section, click a type to display the complete list of task types:
 - **Alerted**

When a task is in an alerted state, you can notify recipients. However, none of the notification recipients (assignees, approvers, owner, initiator, or reviewer) can move the task from an alerted state to an error state; they only receive an FYI notification of the alerted state. The owner can reassign, withdraw, delete, or purge the task, or ask the error assignee to move the task to an error state if the error cannot be resolved. Only the error assignee can move a task from an alerted state to an error state.

You configure the error assignee on the **Assignment** tab of the Configure Assignment dialog under the **Task will go from starting to final participant** icon in the **Assignment and Routing Policy** section. For more information, see [Section 25.3.7.4, "Configuring the Error Assignee."](#)

- **Assign**

When the task is assigned to users or a group. This captures the following actions:

- Task is assigned to a user
- Task is assigned to a new user in a serial workflow
- Task is renewed
- Task is delegated
- Task is reassigned
- Task is escalated
- Information for a task is submitted

- **Complete**

- **Error**

- **Expire**

- **Request Info**

- **Resume**

- **Suspend**

- **Update**

- Task payload is updated
- Task is updated
- Comments are added
- Attachments are added and updated

- **Update Outcome**

- **Withdraw**

- **All Other Actions**

- Any action not covered in the above task types. This includes acquiring a task.

2. Select a task status type.

Notifications can be sent to users involved in the task in various capacities. This includes when the task is assigned to a group, each user in the group is sent a notification if there is no notification endpoint available for the group.

3. In the **Recipient** column, click an entry to display a list of possible recipients for the notification message:

- **Assignees**

The users or groups to whom the task is currently assigned.

- **Initiator**

The user who created the task.

- **Approvers**

The users who have acted on the task up to this point. This applies in a serial participant type in which multiple users have approved the task and a notification must be sent to all of them.

- **Owner**

The task owner

- **Reviewer**

The user who can add comments and attachments to a task.

For more information, see [Section 29.2.5, "How to Configure the Notification Channel Preferences."](#)

25.3.9.2 Editing the Notification Message

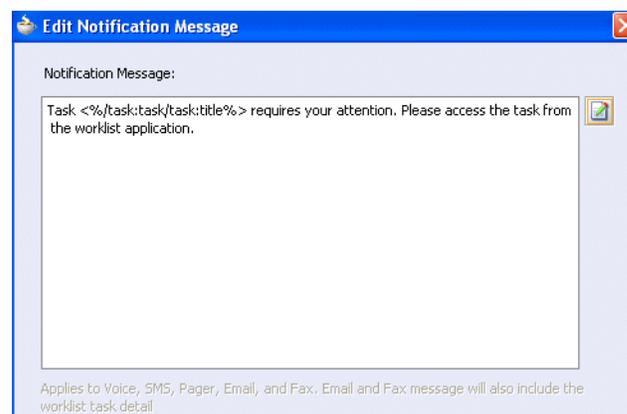
A default notification message is available for delivery to the selected recipient. If you want, you can modify the default message text.

To edit the notification message:

1. In the **Notification Header** column of the **Notification Settings** section, click the **Edit** icon to modify the default notification message.

The Edit Notification Message dialog shown in [Figure 25–59](#) appears.

Figure 25–59 Edit Notification Message Dialog



This message applies to all the supported notification channels: email, voice, instant messaging, and SMS. Email messages can also include the worklist task detail defined in this message. The channel by which the message is delivered is based upon the notification preferences you specify.

2. Modify the message wording as necessary.

3. Click **OK** to return to the Human Task Editor.

For more information about notification preference details, see [Section 29.2, "Notifications from Human Workflow."](#)

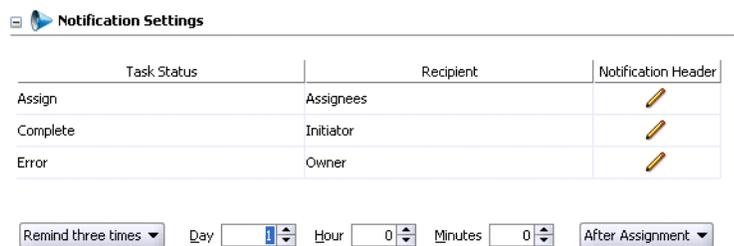
25.3.9.3 Setting Up Reminders

You can send task reminders, which can be based on the time the task was assigned to a user or the expiration time of a task. The number of reminders and the interval between the reminders can also be configured.

To set up reminders:

1. From the **Remind** list in the **Notification Settings** section, select the number of reminders to send.
2. If you selected to remind the assignee one, two, or three times, select the interval between reminders, and whether to send the reminder before or after the assignment. [Figure 25–60](#) provides details.

Figure 25–60 Notification Settings



If you select **Use Due Date** in the **Expiration and Routing Policy** section, the dropdown list at the far right displays an option for selection called **Before Due Date**.

For more information, see [Section 29.2.12, "How to Send Reminders."](#)

25.3.9.4 Changing the Character Set Encoding

Unicode is a universally encoded character set that enables information from any language to be stored using a single character set. Unicode provides a unique code value for every character, regardless of the platform, program, or language. You can use the default setting of UTF-8 or you can specify a character set with a Java class.

To change the character set encoding

1. From the **Encoding** list, select **Specify by Java Class**.
2. Enter the Java class to use.

25.3.9.5 Securing Notifications to Exclude Details

To secure notifications, make messages actionable, and send attachments:

1. Select **Make notifications secure (exclude details)** in the **Notification Settings** section.

If selected, a default notification message is used. There are no HTML worklist task details, attachments, or actionable links in the email. Only the task number is in the message.

For more information, see [Section 29.2.10, "How to Send Secure Notifications."](#)

25.3.9.6 Making Email Messages Actionable

1. Select **Make notification actionable** in the **Notification Settings** section. This action enables you to perform task actions through email.

For more information about additional configuration details, see [Section 29.2.7, "How to Send Actionable Messages"](#).

For more information about configuring outbound and inbound emails, see *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite*.

25.3.9.7 Sending Task Attachments with Email Notifications

1. Select **Send task attachments with email notifications** in the **Notification Settings** section.
2. If you also want to customize the notification headers, select **Custom Notification Headers**.

Custom notification headers are used to specify name and value pairs to identify key fields within the notification. These entries can be used by users to define delivery preferences for their notifications. For example:

You can set **Name** to **ApprovalType** and **value** to **Expense** or **Name** to **Priority** and **value** to **High**.

Users can then specify delivery preferences in Oracle BPM Worklist. These preferences can be based on the contents of the notification.

Note that the rule-based notification service is *only* used to identify the preferred notification channel to use. The address for the preferred channel is still obtained from the identity service.

3. Add name and pair value parameters by name or XPath expression.

For more information about preferences, see [Section 29.2.8, "How to Send Inbound and Outbound Attachments,"](#) [Section 29.2.14, "How to Create Custom Notification Headers,"](#) and [Part VII, "Using Oracle User Messaging Service"](#).

25.3.10 How To Specify Advanced Settings

This section enables you to specify advanced human task features, such as specifying custom escalation rules, custom style sheets for attachments, multilingual settings, callback classes, digital signature policies, access to task content and actions, restricted assignments, task and routing customization in BPEL callbacks, and graphical histories.

[Figure 25–61](#) shows the advanced settings section of the Human Task Editor.

Figure 25–61 Human Task Editor — Advanced Settings Section

 **Advanced Settings**

Specify Escalation Rule.

Specify WordML stylesheet for attachments. 

Specify stylesheet for attachments. 

Specify multilingual settings.

Specify callback class on task status.

Specify workflow signature policy.

Override default access to task content and actions

Specify Restricted Assignment

Allow task and routing customization in BPEL callbacks

Show Complete Graphical History

Table 25–15 describes the sections available.

Table 25–15 Advanced Settings Sections

Section	See...
Specify Escalation Rule	Section 25.3.10.1, "Specifying Escalation Rules"
Specify WordML Stylesheet for attachments	Section 25.3.10.2, "Specifying WordML Style Sheets for Attachments"
Specify stylesheet for attachments	Section 25.3.10.3, "Specifying Style Sheets for Attachments"
Specify multilingual settings	Section 25.3.10.4, "Specifying Multilingual Settings"
Specify callback class on task status	Section 25.3.10.5, "Specifying Callback Classes on Task Status"
Specify workflow signature policy	Section 25.3.10.6, "Specifying a Workflow Signature Policy" Section 25.3.10.7, "Specifying a Certificate Authority"
Override default access to task content	Section 25.3.10.8, "Specifying Access Policies on Task Content"
Specify Restricted Assignment	Section 25.3.10.9, "Specifying Restrictions on Task Assignments"
Allow task and routing customization in BPEL callbacks	Section 25.3.10.10, "Allowing Task and Routing Customization in BPEL Callbacks"
Show Complete Graphical History	Section 25.3.10.11, "Showing the Complete Graphical History"

25.3.10.1 Specifying Escalation Rules

This option allows a custom escalation rule to be plugged in for a particular workflow. For example, to assign the task to a current user’s department manager on task expiration, you can write a custom task escalation function, register it with the workflow service, and use that function in task definitions.

The default escalation rule is to assign a task to the manager of the current user. To add a new escalation rule, follow these steps.

To specify escalation rules:

1. Implement interface `oracle.bpel.services.workflow.assignment.dynamic.IDynamicTaskEscalationFunction`. This implementation has to be available in the class path for the server.
2. Log in to Oracle Enterprise Manager Fusion Middleware Control Console.
3. Expand the **SOA** folder in the navigator.
4. Right-click **soa-infra**, and select **SOA Administration > Workflow Task Service Properties**.

The Workflow Task Service Properties page appears.

5. Add a new function. For example:
 - Function name: `Department_supervisor`
 - Classpath:
`oracle.bpel.services.workflow.assignment.dynamic.patterns.DepartmentSupervisor`
 - Function parameter name
 - Function parameter value
6. In the **Specify Escalation Rule** field of the **Advanced Settings** section, enter the function name as defined in the Workflow Task Service Properties page for the escalation rule.

For more information, see [Section 29.3.3, "Custom Escalation Function."](#)

25.3.10.2 Specifying WordML Style Sheets for Attachments

This option allows for the dynamic creation of Microsoft Word documents for sending them as email attachments using a WordML XSLT style sheet. The XSLT style sheet is applied on the task document.

To specify WordML style sheets for attachments:

1. In the **Specify WordML stylesheet for attachments** field of the **Advanced Settings** section, click the **Search** icon to select a WordML style sheet as an attachment.

25.3.10.3 Specifying Style Sheets for Attachments

This option allows creation of email attachments using an XSLT style sheet. The XSLT style sheet is applied on the task document.

To specify style sheets for attachments:

1. In the **Specify stylesheet for attachments** field of the **Advanced Settings** section, click the **Search** icon to select a style sheet as an attachment.

25.3.10.4 Specifying Multilingual Settings

You can specify resource bundles for displaying task details in different languages in Oracle BPM Worklist. Resource bundles are supported for the following task details:

- Displaying the value for task outcomes in plain text or with the message (key) format

- Displaying the XML element and attributes names in the payload display of Oracle BPM Worklist. The key name in the resource bundle must be the same as the name of the XML element and attributes for internationalization of XML element names in Oracle BPM Worklist.
- Making email notification messages available in different languages. At runtime, specify the XPath extension function `hwf:getTaskResourceBundleString(taskId, key, locale?)` to obtain the internationalized string from the specified resource bundle. The locale of the notification recipient can be retrieved with the function `hwf:getNotificationProperty(propertyName)`.

Resource bundles can also simply be property files. For example, a resource bundle that configures a display name for task outcomes can look as follows:

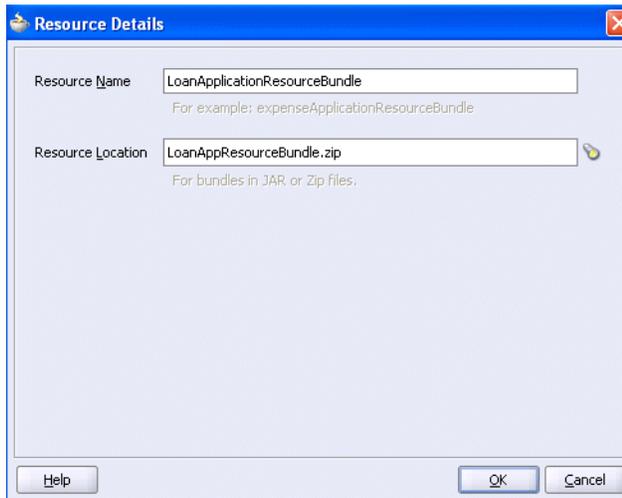
- APPROVE=Approve
- REJECT=Reject

To specify multilingual settings:

1. In the **Specify multilingual settings** field of the **Advanced Settings** section, click **Configure Resource**.

The Resource Details dialog shown in [Figure 25–62](#) appears.

Figure 25–62 Resource Details Dialog



2. In the **Resource Name** field, enter the name of the resource used in the resource bundle. This should be a `.properties` based resource bundle file.
3. In the **Resource Location** field, click the **Search** icon to select the JAR or ZIP resource bundle file to use. The resource bundle is part of your system archive (SAR) file.

If the resource bundle is outside of the composite project, you are prompted to place a local copy in `SCA-INF/lib`.

If the resource bundle file is not in the composite class loader (directly under `SCA-INF/classes` or in a JAR file in `SCA-INF/lib`), you must specify its location. For example, if the resource bundle is accessible from a location outside of the composite class loader (such as an HTTP location such as

`http://host:port/bundleApp/taskBundles.jar`), then this location must be specified in this field.

4. Click **OK** to return to the Human Task Editor.

For more information, see [Section 29.2.6, "How to Configure Notification Messages in Different Languages."](#)

25.3.10.5 Specifying Callback Classes on Task Status

You can register callbacks for the workflow service to call when a particular stage is reached during the lifecycle of a task. Two types of callbacks are supported:

- **Java callbacks:** The callback class must implement the interface `oracle.bpel.services.workflow.task.IRoutingSlipCallback`. Make the callback class available in the class path of the server.
- **Business event callbacks:** You can have business events raised when the state of a human task changes. You do not need to develop and register a Java class. The caller implements the callback using a mediator service component to subscribe to the applicable business event to be informed of the current state of an approval transaction.

To specify callback classes on task status:

1. In the **Specify callback class on task status** field of the **Advanced Settings** section, click **Configure Callbacks**.
2. Click the **Add** icon to add a callback to the table. A callback named **OnAssigned** is automatically added to the **Callback** column.
3. Click **OnAssigned** to display a list of additional callback values to select for this column.

The following callbacks are available:

- **OnCompleted**

Select if the callback class must finally be called when the task is completed and control is about to be passed to the initiator (such as the BPEL process initiating the task).

- **OnAssigned**

Select if the callback class must be called on any assignment change, including standard routing, reassignment, delegation, escalation, and so on. If a callback is required when a task has an outcome update (that is, one of the approvers in a chain approves or rejects the task), this option must be selected.

- **OnUpdated**

Select if the callback class must be called on any update (including payload, comments, attachment, priority, and so on).

- **OnSubtaskUpdated**

Select if the callback class must be called on any update (including payload, comments, attachment, priority, and so on) on a subtask (one of the tasks in a parallel and parallel scenario).

- **OnStageCompleted**

Select if the callback class must be called to enable business event callbacks in a human workflow task. When the event is raised, it contains the name of the

completed stage, the outcome for the completed stage, and a snapshot of the task when the callback is invoked.

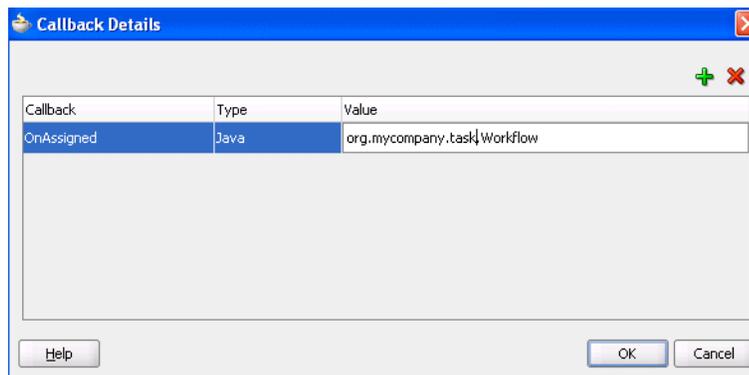
4. See the following section based on the type callback to perform.
 - [Section 25.3.10.5.1, "Specifying Java Callbacks"](#)
 - [Section 25.3.10.5.2, "Specifying Business Event Callbacks"](#)

25.3.10.5.1 Specifying Java Callbacks

To specify Java callbacks:

1. In the **Type** column, click **Java**.
2. In the **Value** column, click the empty field to enter a value. This value is the complete class name of the Java class that implements `oracle.bpel.services.workflow.task.IRoutingSlipCallback`. [Figure 25–63](#) provides details.

Figure 25–63 *Callback Details Dialog with Java Selected*

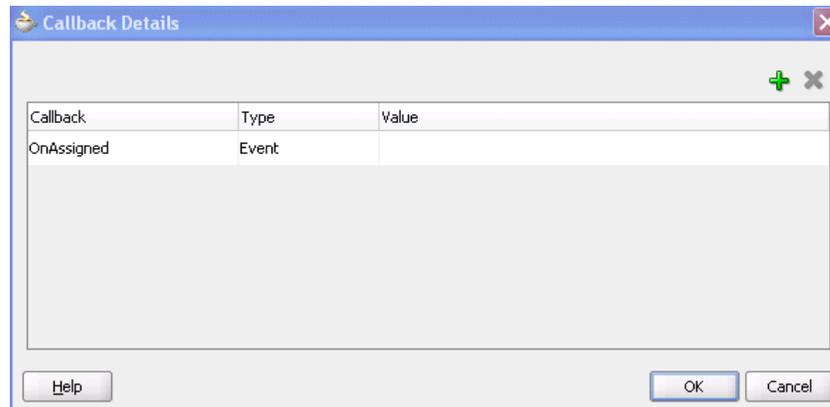


3. Click **OK**.

25.3.10.5.2 Specifying Business Event Callbacks

To specify business event callbacks:

1. In the **Type** column, click **Events**. This action disables the **Value** column, as shown in [Figure 25–64](#). Each callback, such as **OnAssigned**, corresponds to a business event point. When a business event is fired, the event details contain the task object and a set of properties that are populated based on the context of the event being fired.

Figure 25–64 Callback Details Dialog with Business Events Selected

A preseeded, static event definition language (EDL) file (*JDev_Home\jdeveloper\integration\seed\soa\shared\workflow\HumanTaskEvent.edl*) provides the list of available business events to which to subscribe. These business events correspond to the callbacks you select in the Callback Details dialog. You must now create a mediator service component in which you reference the EDL file and subscribe to the appropriate business event.

Note: A file-based MDS connection is required so that the EDL file can be located. The location for the file-based MDS is *JDev_Home\jdeveloper\integration\seed*.

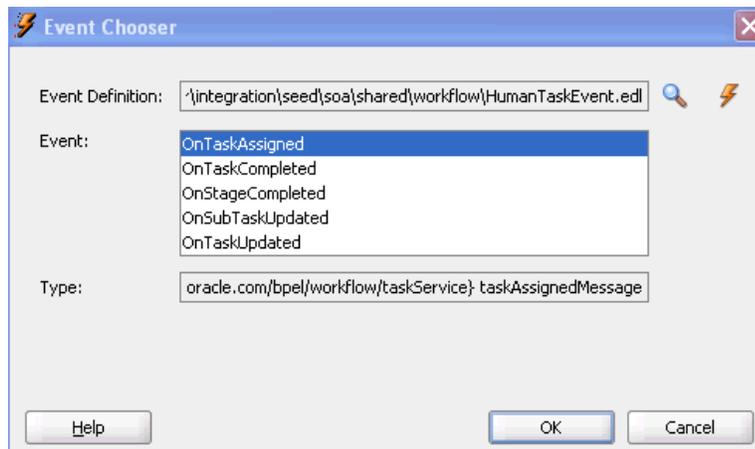
2. Create an Oracle Mediator service component in the same or a different SOA composite application that can subscribe to the event.
3. In the **Template** list during Oracle Mediator creation, select **Subscribe to Events**.
4. Click the **Add** icon to subscribe to a new event.
5. To the right of the **Event Definition** field, click the **Browse** icon to select the EDL file.

The SOA Resource Browser dialog appears.

6. Select the previously created file-based MDS connection.
7. From the list at the top, select **Resource Palette**.
8. Select **SOA > Shared > Workflow > HumanTaskEvent.edl**.
9. Click **OK**.

The Event Chooser is now populated with EDL file business events available for selection.

10. In the **Event** field, select the event to which to subscribe. [Figure 25–65](#) provides details.

Figure 25–65 Event Callbacks

You can have multiple human tasks available for subscribing to the event. For example, assume you have the following:

- Configured a human task named TaskA to subscribe to the event (for example, **OnAssigned**)
- Configured a human task named TaskB to subscribe to the same event

To distinguish between events for TaskA and TaskB and ensure that an event is processed only by the intended Oracle Mediator, you can add a static routing filter:

```
xpath20:compare (med:getComponentName () , 'TaskA')
```

This only invokes this routing when the sending component is TaskA.

11. If the EDL file was *not* selected from the file-based MDS connection, accept to import the dependent XSD files when prompted, and click **OK**. If the EDL file was selected from the file-based MDS connection, you are not prompted.

The Oracle Mediator service component is now populated with the business event to which to subscribe. You can also subscribe to other business events defined in the same EDL file now or at a later time.

See the following documentation for additional details about business events and callbacks:

- [Chapter 44, "Using Business Events and the Event Delivery Network"](#) for specific details about business events
- Sample workflow-116-WorkflowEventCallback, which is available from the Oracle Technology Network:

http://www.oracle.com/technology/sample_code/products/hwf

25.3.10.6 Specifying a Workflow Signature Policy

Digital signatures provide a mechanism for the nonrepudiation of digitally-signed human tasks. This ability to mandate that a participant acting on a task signs the details and their action before the task is updated ensures that they cannot repudiate it later.

Note: If digital signatures are enabled for a task, actionable emails are not sent during runtime. This is the case even if actionable emails are enabled during design-time.

To specify a workflow signature policy:

1. In the **Specify workflow signature policy** field of the **Advanced Settings** section, click **Configure Policy**.
2. Specify the signature policy for task participants to use:
 - **No signature required**
Participants can send and act upon tasks without providing a signature. This is the default policy.
 - **Password required**
Participants specify a signature before sending tasks to the next participant. Participants must reenter their password while acting on a task. The password is used to generate the digital signature. A digital signature authenticates the identity of the message sender or document signer. This ensures that the original content of the sent message is unchanged.
 - **Digital certificate required**
Participants must possess a digital certificate for the nonrepudiation of digitally-signed human tasks. A digital certificate establishes the participant's credentials. It is issued by a certification authority (CA). It contains the following:
 - Your name
 - A serial number
 - Expiration dates
 - A copy of the certificate holder's public key (used for encrypting messages and digital signatures)
 - Digital signature of the certificate-issuing authority so that message authenticity can be established

The CA names and CA CRL and URLs of the issuing authorities must be configured separately.
3. Click **OK**.

For more information, see [Section 29.1.10, "Evidence Store Service and Digital Signatures."](#)

25.3.10.7 Specifying a Certificate Authority

To use digital signatures, you must specify CAs you consider trustworthy in the `workflow-config.xml` file. Only certificates issued from such CAs are considered valid by human workflow.

To specify a certificate authority:

1. Edit the `workflow-config.xml` file to include trustworthy CAs.

```
<trustedCAList>
  <trustedCA CName="CN=VeriSign Class 1 Individual Subscriber CA
-G2,OU=Persona Not Validated,OU=Terms of use at
```

```

https://www.verisign.com/rpa(c)05,OU=VeriSign Trust Network,O=VeriSign\,
Inc.,C=US"
CAURL="http://IndC1DigitalID-crl.verisign.com/IndC1DigitalID.crl"/>
  <trustedCA CName="CN=Thawte Personal Freemail Issuing CA,O=Thawte
Consulting (Pty) Ltd.,C=ZA"
CAURL="http://crl.thawte.com/ThawtePersonalFreemailIssuingCA.crl"/>
</trustedCAList>

```

Note that these CAs are used for indicative purposes only. You must validate these values before using them.

25.3.10.8 Specifying Access Policies on Task Content

You can specify access rules that determine the parts of a task that participants can view and update. Access rules are enforced by the workflow service by applying rules on the task object during the retrieval and update of the task.

Note: Task content access rules and task actions access rules exist independently of one another.

25.3.10.8.1 Introduction to Access Rules Access rules are computed based on the following details:

- Any attribute configured with access rules declines any permissions for roles not configured against it. For example, assume you configure the payload to be read by assignees. This action enables *only* assignees and nobody else to have read permissions. No one, including assignees, has write permissions.
- Any attribute not configured with access rules has *all* permissions.
- If any payload message attribute is configured with access rules, any configurations for the payload itself are ignored due to potential conflicts. In this case, the returned map by the API does not contain any entry for the payload. Write permissions automatically provide read permissions.
- If only a subset of message attributes is configured with access rules, all message attributes not involved have all permissions.
- Only comments and attachments have add permissions.
- Write permissions on certain attributes are meaningless. For example, write permissions on history do not grant or decline any privileges on history.
- The following date attributes are configured as one in the Human Task Editor. The map returned by `TaskMetadataService.getVisibilityRules()` contains one key for each. Similarly, if the participant does not have read permissions on DATES, the task does not contain any of the following task attributes:
 - START_DATE
 - END_DATE
 - ASSIGNED_DATE
 - SYSTEM_END_DATE
 - CREATED_DATE
 - EXPIRATION_DATE
 - ALL_UPDATED_DATE

- The following assignee attributes are configured as one in the Human Task Editor. The map returned by `TaskMetadataService.getVisibilityRules()` contains one key for each of the following. Similarly, if the participant does not have read permissions on ASSIGNEES, the task does not contain any of the following task attributes:
 - ASSIGNEES
 - ASSIGNEE_USERS
 - ASSIGNEE_GROUPS
 - ACQUIRED_BY
- Flex fields do not have individual representation in the map returned by `TaskMetadataService.getVisibilityRules()`.
- All message attributes in the map returned by `TaskMetadataService.getVisibilityRules()` are prefixed by `ITaskMetadataService.TASK_VISIBILITY_ATTRIBUTE_PAYLOAD_MESSAGE_ATTR_PREFIX (PAYLOAD)`.

An application can also create pages to display or not display task attributes based on the access rules. This can be achieved by retrieving a participant's access rules by calling the API on `oracle.bpel.services.workflow.metadata.ITaskMetadataService`. [Example 25-1](#) provides details.

Example 25-1 API Call

```
public Map<String, IPrivilege> getTaskVisibilityRules(IWorkflowContext context,
                                                    String taskId)
    throws TaskMetadataServiceException;
```

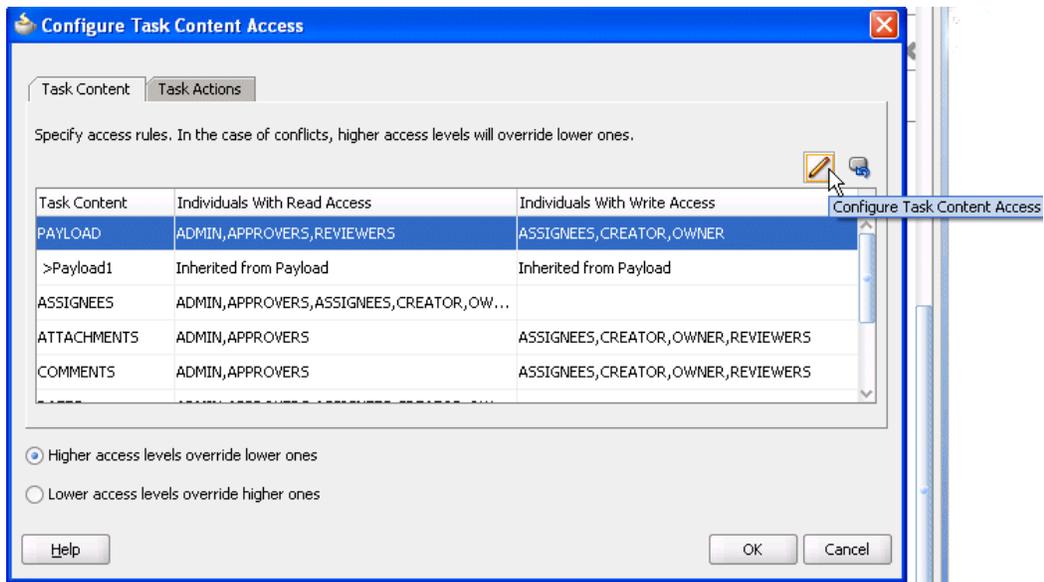
For more information about this method, see *Oracle Fusion Middleware Workflow Services Java API Reference for Oracle BPEL Process Manager*.

25.3.10.8.2 Specifying User Privileges for Acting on Task Content You can specify the privileges that specific users (such as the task creator or owner) have for acting on specific task content (such as a payload).

To specify user privileges for acting on task content:

1. In the **Override default access to task content and actions** field of the **Advanced Settings** section, click **Configure Visibility**.
The Configure Task Content Access dialog appears.
2. Select the task content for which to specify access privileges (for this example, **PAYLOAD**).
3. Click the **Edit** icon, as shown in [Figure 25-66](#).

Figure 25–66 Configure Task Content Access



The Configure Task Content Access dialog appears.

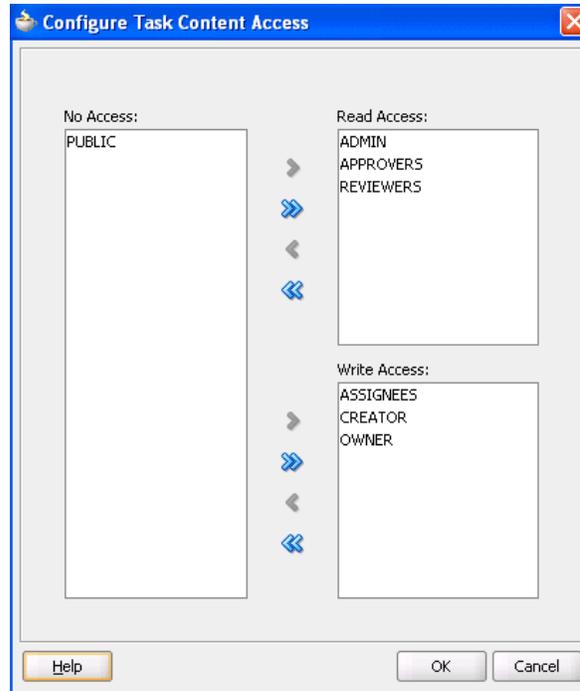
4. Assign privileges (read, write, or no access) to users to act upon task content. Note that a user cannot be assigned a privilege above their highest level. For example, an **ADMIN** user cannot be assigned write access on the **PAYLOAD** task content. If you attempt this assignment, you receive an error. [Table 25–16](#) shows the maximum privilege each user has on task content.

Table 25–16 Highest Privilege Levels for Users of Task Content

Task Content	Individual with Read Access	Individual with Write Access
PAYLOAD	ADMIN, APPROVERS, REVIEWERS	ASSIGNEES, CREATOR, OWNER
ASSIGNEES	ASSIGNEES, CREATOR, OWNER, ADMIN, APPROVERS, REVIEWERS	
ATTACHMENTS	ADMIN, APPROVERS	ASSIGNEES, CREATOR, OWNER, REVIEWERS
COMMENTS	ADMIN, APPROVERS	ASSIGNEES, CREATOR, OWNER, REVIEWERS
DATES	ASSIGNEES, CREATOR, OWNER, ADMIN, APPROVERS, REVIEWERS	
FLEXFIELDS	ADMIN, APPROVERS, REVIEWERS	ASSIGNEES, CREATOR, OWNER
HISTORY	ASSIGNEES, CREATOR, OWNER, ADMIN, APPROVERS, REVIEWERS	--
REVIEWERS	ASSIGNEES, CREATOR, OWNER, ADMIN, APPROVERS, REVIEWERS	--
Payload elements	Inherited from payload	Inherited from payload

For example, if you accept the default setting of **ASSIGNEES**, **CREATOR**, and **OWNER** with write access, **ADMIN**, **APPROVERS**, and **REVIEWERS** with read access, and **PUBLIC** with no access to the **PAYLOAD** task content, the dialog appears as shown in [Figure 25–67](#):

Figure 25–67 *Configure Task Content Access*



5. Click **OK**.

You are returned to the Configure Task Content Access dialog.

The **Higher access levels override lower ones** and **Lower access levels override higher ones** radio buttons at the bottom of the dialog address scenarios in which:

- A user plays more than one role in a task, such as owner, creator, and assignee.
- A user belongs to several groups.
- A user is granted several application roles, and different access levels are defined for each of them.

6. Select an option appropriate to your environment. [Table 25–17](#) provides details.

Table 25–17 *Access Rule Modes*

Option	Select...
Higher access levels override lower ones	<p>To permit a user to perform an action on a task or make changes to an attribute of a task if one of the following is true:</p> <ul style="list-style-type: none"> ■ They perform at least one role where it is permitted. ■ They are a member of any participant type where it is permitted. ■ They are a member of any one group where it is permitted. ■ They are granted any one application role where it is permitted.

Table 25–17 (Cont.) Access Rule Modes

Option	Select...
Lower access levels override higher ones	<p>To prevent a user from performing an action on a task or making changes to an attribute of a task if one of the following is true:</p> <ul style="list-style-type: none"> ■ They perform any role where it is not permitted. ■ They are a member of any participant type where it is not permitted. ■ They are a member of any one group where it is not permitted. ■ They are granted any one application role where it is not permitted.

Note: Access rules are always applied on top of what the system permits, depending on who is performing the action and the current state of the task.

25.3.10.8.3 Specifying Actions for Acting Upon Tasks You can specify the actions (either access or no access) that specific users (such as the task creator or owner) have for acting on the task content (such as a payload) that you specified in the Configure Task Content Access dialog shown in [Figure 25–67](#).

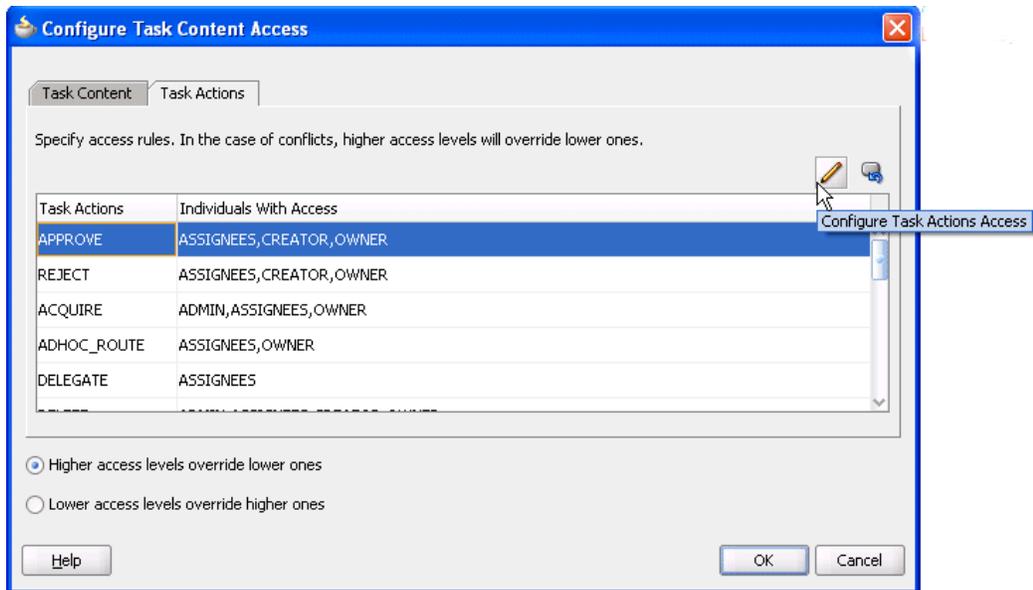
To specify actions for acting upon tasks:

1. In the **Override default access to task content and actions** field of the **Advanced Settings** section, click **Configure Visibility**.

The Configure Task Content Access dialog appears.

2. Click the **Task Actions** tab.
3. Select the task action for which to specify users.
4. Click the **Edit** icon shown in [Figure 25–68](#).

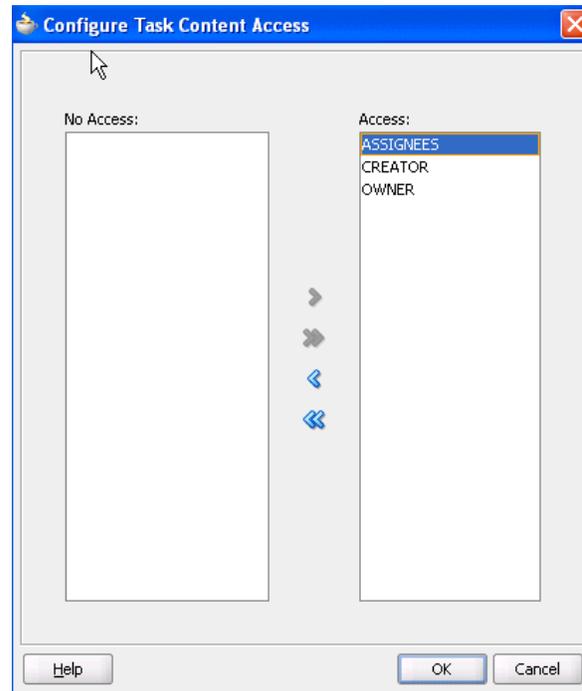
Figure 25–68 Selection of Add Action Access Rule



The Configure Task Content Access dialog appears.

5. Select if participants can or cannot perform the selected actions, as shown in Figure 25–69.

Figure 25–69 *Configure Task Content Access Dialog*



6. Click **OK**.

You are returned to the Configure Task Content Access dialog.

The **Higher access levels override lower ones** and **Lower access levels override higher ones** radio buttons display at the bottom of the dialog.

7. See Step 6 on page 25-73 for descriptions of how to use these radio buttons.

25.3.10.9 Specifying Restrictions on Task Assignments

You can restrict the users to which a task can be reassigned or routed through a callback class.

To specify restrictions on task assignments:

1. In the **Specify Restricted Assignment** field of the **Advanced Settings** section, click **Configure Restricted Assignments**.
The Configure Restricted Assignment dialog appears.
2. Enter the class name. The class must implement the `oracle.bpel.services.workflow.task.IRestrictedAssignmentCallback` interface.
3. Click the **Add** icon to add name and value pairs for the property map passed to invoke the callback.
4. Click **OK**.

25.3.10.10 Allowing Task and Routing Customization in BPEL Callbacks

In general, the BPEL process calls into the workflow component to assign tasks to users. When the workflow is complete, the human workflow service calls back into the BPEL process. However, if you want fine-grained callbacks (for example, `onTaskUpdate` or `onTaskEscalated`) to be sent to the BPEL process, you must use this option.

Make sure to manually refresh the BPEL diagram for this callback setting.

To allow task and routing customization in BPEL callbacks:

1. Select the **Allow Task and Routing Customization in BPEL Callbacks** checkbox.
2. Return to Oracle BPEL Designer.
3. Open the task activity dialog.
4. Click **OK**.

This creates the while, pick, and `onMessage` branch of a pick activity for BPEL callback customization inside the task scope activity.

25.3.10.11 Showing the Complete Graphical History

You can view either the complete workflow history (across routing participants) or only the task history in Oracle BPM Worklist.

1. In the **Advanced Settings** section, click **Show Complete Graphical History**.

25.3.11 How to Specify Annotations

Annotations are used to label different attributes of the task definition. This functionality is used with Oracle Business Process Analysis in which a model designed by a business user is transformed to a complete Oracle BPEL Process Manager human task definition. During translation, any comments the business user has included for the developer or IT user are stored as annotations. These annotations are then used by the developer or IT user to complete the modeling process.

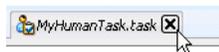
25.3.12 How to Exit the Human Task Editor and Save Your Changes

You can save your human task changes at any time. The task can be re-edited at a later time by clicking the metadata task configuration `.task` file in the Application Navigator.

To exit the Human Task Editor and save your changes:

1. From the **File** main menu, select **Save** or click the **X** sign shown in [Figure 25–70](#) to close the `.task` metadata task configuration file.

Figure 25–70 File Closure



2. If you click the **X** sign, select **Yes** when prompted to save your changes.

25.4 Associating the Human Task Service Component with a BPEL Process

If you want to associate the human task service component created in the SOA Composite Editor with a BPEL process, follow these instructions. When association is complete, a task service partner link is created in Oracle BPEL Designer. The task service exposes the operations required to act on a task.

For more information about creating a human task, see [Section 25.3, "Creating the Human Task Definition with the Human Task Editor."](#)

25.4.1 How to Associate a Human Task with a BPEL Process

There are two ways to associate a human task component with a BPEL process:

- If you have created a human task component in the SOA composite application, drag a human task activity into the BPEL process in Oracle BPEL Designer. Then, select the existing human task component from the **Task Definition** list of the Create Human Task dialog. You can then specify the task title, initiator, parameter values, and other values.
- If you have not created a human task component, drag the human task activity into the BPEL process in Oracle BPEL Designer. Then, click the **Add** icon to the right of the **Task Definition** list in the Create Human Task dialog. This action enables you to specify the name of the new human task component, the parameters, and the outcomes. The Human Task Editor then opens for you to design the remaining task metadata. After design completion, close the Human Task Editor.

To associate a human task with a BPEL process:

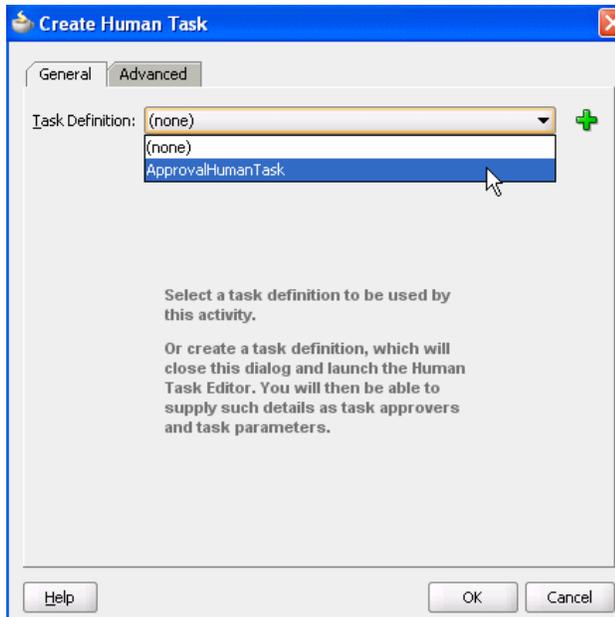
1. Go to the SOA Composite Editor.
2. Double-click the BPEL process service component with which to associate the .task file of the human task service component.
3. From the Component Palette, select **BPEL**.
4. Expand **BPEL Activities**.
5. Drag a new **Human Task** activity into the BPEL process.

The Create Human Task dialog appears.

Note: When you first drag this activity into Oracle JDeveloper, the dialog is named *Create Human Task*. After you finish specifying details on this dialog and click **OK**, the name of this dialog changes to simply *Human Task*.

6. From the **Task Definition** list of the **General** tab, select the human task, as shown in [Figure 25-71](#).

Figure 25–71 Task Definition List Selection



The .task file of the human task service component is associated with the BPEL process.

Note: After you complete association of your human task activity with a BPEL process and close the Create Human Task dialog, you can always re-access this dialog by double-clicking the human task activity in Oracle BPEL Designer.

25.4.2 What You May Need to Know About Deleting a Wire Between a Human Task Service Component and a BPEL Process

If you delete the wire between a BPEL process and the human task service component that it invokes, the invoke activity of the human workflow is deleted from the BPEL process. However, the **taskSwitch** switch activity for taking action (contains the approve, reject, and otherwise task outcomes) is still there. This is by design for the following reasons:

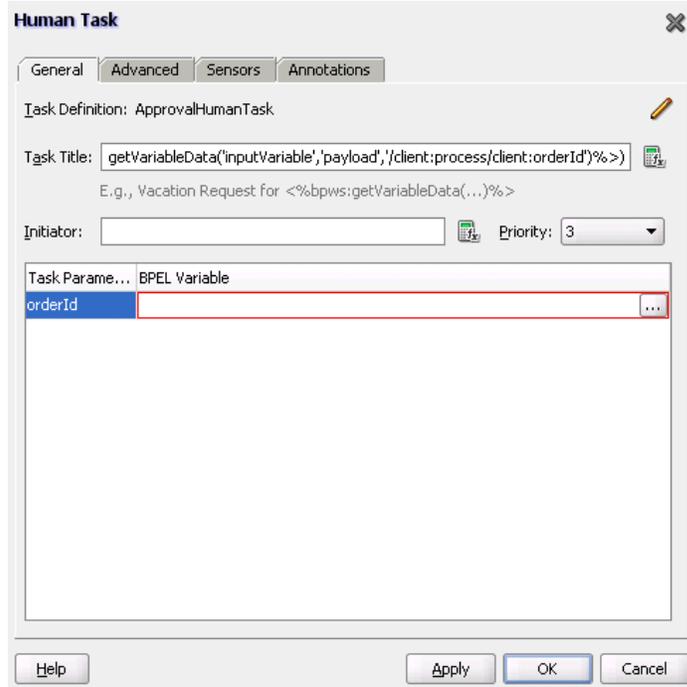
- The switch activity contains user-entered BPEL code.
- The switch can be reused if the intention for deleting the wire is only to point to another human task.
- Deleting the switch is a single-step action.

If you then drag and drop a human task service component into the BPEL process to use the same **taskSwitch** switch activity, a new **taskSwitch** switch activity is created. You then have two switch activities in the BPEL process with the same name. To determine which one to delete, you must go into the approve, reject, and otherwise task outcomes of the **taskSwitch** switch activities to determine which is the older, modified switch and which is the newer switch.

25.4.3 How to Define the Human Task Activity Title, Initiator, Priority, and Parameter Variables

Figure 25–72 shows the **General** tab that displays after you select the human task.

Figure 25–72 Human Task — General Tab (After Selection)



The **General** tab of the Human Task activity enables you to perform the tasks shown in Table 25–18:

Table 25–18 Human Task - General Tab

For this Field...	See...
Task Title	Section 25.4.3.1, "Specifying the Task Title"
Initiator	Section 25.4.3.2, "Specifying the Task Initiator and Task Priority"
Priority	
Task Parameters	Section 25.4.3.3, "Specifying Task Parameters"

25.4.3.1 Specifying the Task Title

The title displays the task in Oracle BPM Worklist during runtime. This is a mandatory field. Your entry in this field overrides the task title you entered in the **Title** field of the Human Task Editor described in [Section 25.3.4.1, "Specifying a Task Title."](#)

To specify the task title:

- In the **Task Title** field of the **General** tab, enter the task title through one of the following methods:
 - Enter the title manually.
 - Click the icon to the right of the field to display the Expression Builder dialog to dynamically create the title.

You can also combine static text and dynamic expressions in the same title. To include dynamic text, place your cursor at the appropriate point in the text and click the icon on the right to invoke the Expression Builder dialog.

25.4.3.2 Specifying the Task Initiator and Task Priority

You can specify a task initiator. The initiator is the user who initiates a task. The initiator can view their created tasks from Oracle BPM Worklist and perform specific tasks, such as withdrawing or suspending a task.

To specify the task initiator and task priority:

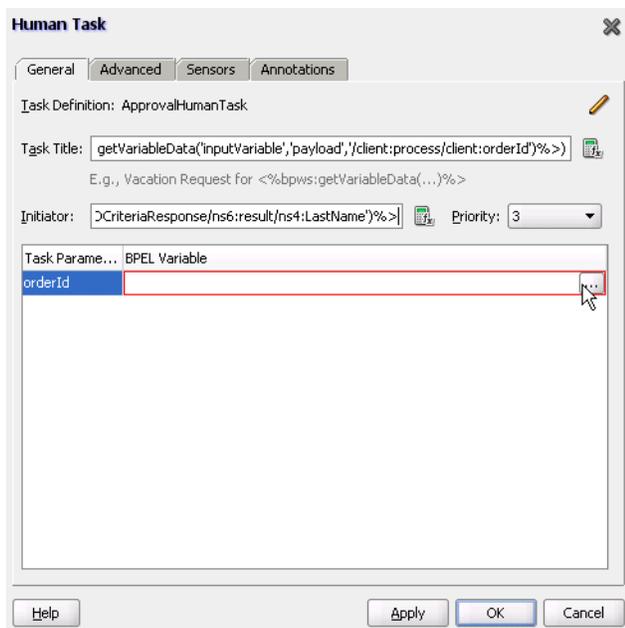
1. To the right of the **Initiator** field of the **General** tab, enter the initiator (for example, `jcooper`) or click the icon to display the Expression Builder dialog for dynamically specifying an initiator. This field is optional. If not specified, the initiator defaults to the task owner specified on the **Advanced** tab of the Human Task dialog. The initiator defaults to `bpeladmin` if a task owner is also not specified.
2. From the **Priority** list, select a priority value between **1** (the highest) and **5**. This field is provided for user reference and does not make this task a higher priority during runtime. The priority can be used to sort tasks in Oracle BPM Worklist. This priority value overrides the priority value you select in the **Priority** list of the Human Task Editor.

For more information about specifying the priority in the Human Task Editor, see [Section 25.3.4.1, "Specifying a Task Title."](#)

25.4.3.3 Specifying Task Parameters

The task parameter table shown in [Figure 25–73](#) displays a list of task parameters after you complete the **Task Title** and **Initiator** fields.

Figure 25–73 Task Parameter Table



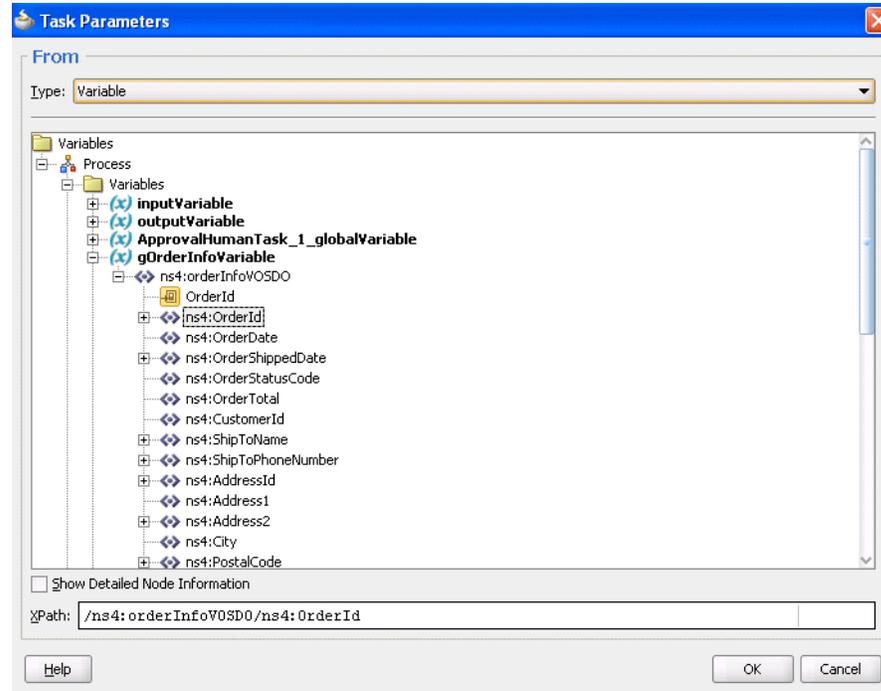
To specify task parameters:

1. In the **BPEL Variable** column, double-click the **dots** 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 Oracle BPM Worklist, you do not need to map the corresponding variables.

The Task Parameters dialog appears.

2. Expand the **Variables** tree shown in [Figure 25–74](#) and select the appropriate task variable.

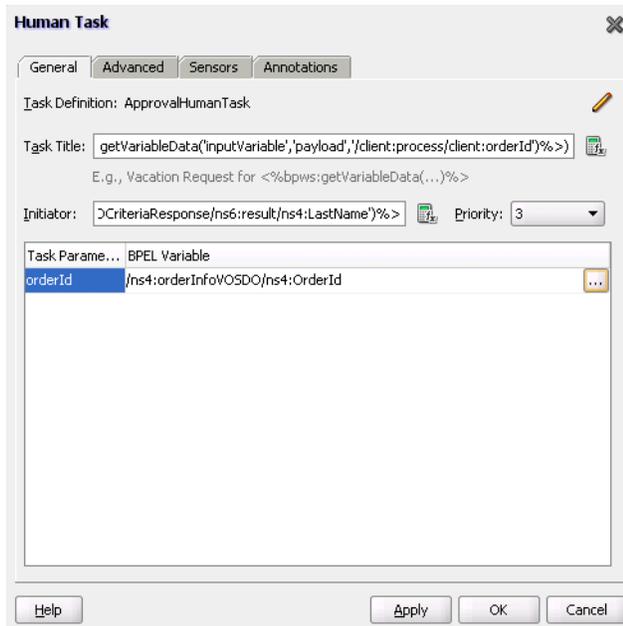
Figure 25–74 Variables Tree



3. Click **OK**.

The Human Task dialog shown in [Figure 25–75](#) appears as follows.

Figure 25–75 Human Task Dialog

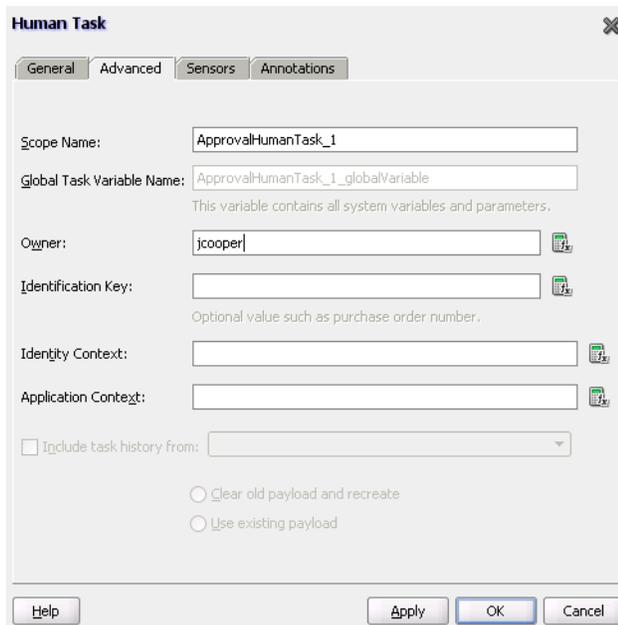


4. Click **OK**.
5. If you want to define advanced features for the human task activity, click the **Advanced** tab and go to [Section 25.4.4, "How to Define the Human Task Activity Advanced Features."](#) Otherwise, click **OK** to close the Human Task dialog.

25.4.4 How to Define the Human Task Activity Advanced Features

Figure 25–76 shows the **Advanced** tab.

Figure 25–76 Create Human Task — Advanced Tab



The **Advanced** tab of the Human Task activity enables you to perform the tasks shown in [Table 25–19](#):

Table 25–19 Human Task - Advanced Tab

For this Field...	See...
Scope Name	Section 25.4.4.1, "Specifying a Scope Name and a Global Task Variable Name"
Global Task Variable Name	
Owner	Section 25.4.4.2, "Specifying a Task Owner"
Identification Key	Section 25.4.4.3, "Specifying an Identification Key"
Identity Context	Section 25.4.4.4, "Specifying an Identity Context"
Application Context	Section 25.4.4.5, "Specifying an Application Context"
Include task history from	Section 25.4.4.6, "Including the Task History of Other Human Tasks"

25.4.4.1 Specifying a Scope Name and a Global Task Variable Name

You are automatically provided with default scope and global task variable names during human task activity creation. However, you can specify custom names that are used to name the scope and global variable during human task activity creation.

To specify a scope name and a global task variable name:

1. In the **Scope Name** field of the **Advanced** tab, enter the name for the BPEL scope to be generated.

This BPEL scope encapsulates the entire interaction with the workflow service and BPEL variable manipulation.

2. In the **Global Task Variable Name** field of the **Advanced** tab, enter the global task variable name.

This is the name of the BPEL task variable used for the workflow interaction.

25.4.4.2 Specifying a Task Owner

The task owner can view tasks belonging to business processes they own and perform operations on behalf of any of the task assignees. Additionally, the owner can also reassign, withdraw, or escalate tasks.

If you do not specify a task initiator on the **General** tab of the Human Task dialog, it defaults to the owner specified here.

To specify a task owner:

1. In the **Owner** field of the **Advanced** tab, enter the task owner name or click the icon to the right to use the Expression Builder to dynamically specify the owner of this task.

25.4.4.3 Specifying an Identification Key

The identification key can be used as a user-defined ID for the task. For example, if the task is meant for approving a purchase order, the purchase order ID can be set as the identification key of the task. Tasks can be searched from Oracle BPM Worklist using the identification key. This attribute has no default value.

To specify an identification key:

1. In the **Identification Key** field of the **Advanced** tab, enter an optional identification key value.

25.4.4.4 Specifying an Identity Context

The identity realm name is used for the task when multiple realms are configured. You cannot have assignees from multiple realms working on the same task. This field is required if you are using multiple realms.

To specify an identity context

1. In the **Identity Context** field of the **Advanced** tab, enter a value.

25.4.4.5 Specifying an Application Context

The stripe name of the application contains the application roles used in the task.

To specify an application context

1. In the **Application Context** field of the **Advanced** tab, enter a value.

25.4.4.6 Including the Task History of Other Human Tasks

This feature enables one human task to be continued with another human task. There are many scenarios in which you have related tasks in a single BPEL process. For example, assume you have a procurement process to obtain a manager's approval for a computer, then several BPEL activities in between, and then another task for the IT department to buy the computer. The participant of the second task may want to see the approval history, comments, and attachments created when the manager approved the purchase. You can link these different tasks in the BPEL process by chaining the second task to the first task with this option.

For chained tasks, the title of the new task cannot be set from the task metadata (.task file). For example, assume existing TaskA is chained with new task TaskB, and TaskB has a new title set in the Human Task Editor; this title is *not* recognized. Therefore, if the chained task requires a different title, it must be set in the task instance before calling the task service `reinitiate` operation. If a BPEL process is initiating the tasks, set the task title before the workflow service APIs are called. If a Java program is calling the workflow APIs programatically, then it must set the title.

To include the task history of other tasks:

1. Select the **Include task history from** checkbox of the **Advanced** tab to extend a previous workflow task in the BPEL process. Selecting this checkbox includes the task history, comments, and attachments from the previous task. This provides you with a complete end-to-end audit trail.

When a human task is continued with another human task, the following information is carried over to the new workflow:

- Task payload and the changes made to the payload in the previous workflow
- Task history
- Comments added to the task in the previous workflow
- Attachments added to the task in the previous workflow

In the **Include task history from** list, all existing workflows are listed.

2. Select a particular human task to extend (continue) the selected human task.

For example, a hiring process is used to hire new employees. Each interviewer votes to hire or not hire a candidate. If 75% of the votes are to hire, then the candidate is hired; otherwise, the candidate is rejected. If the candidate is to be hired, an entry in the HR database is created and the human resources contact completes the hiring process. The HR contact also must see the interviewers and the comments they made about the candidate. This process can be modeled using a parallel participant type for the hiring. If the candidate is hired, a database adapter is used to create the entry in the HR database. After this action, a simple workflow can include the task history from the parallel participant type so that the hiring request, history, and interviewer comments are carried over. This simple workflow is assigned to the HR contact.

3. Select a payload to use:

- **Clear old payload and recreate**

This option is applicable when the payload attributes in the XML files of the human tasks involved in this extended workflow are different. For example, the payload attribute for the human task whose history you are including has three extra attributes than the payload of the other human task.

- **Use existing payload**

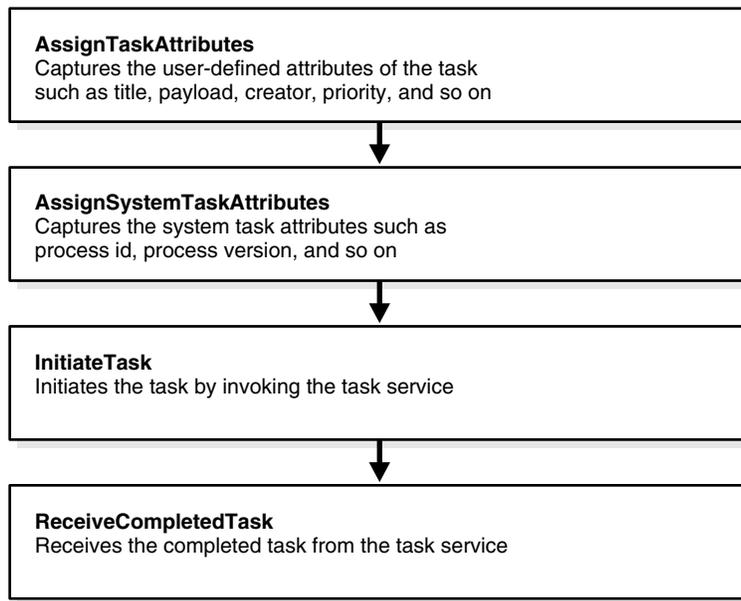
This option is applicable when the payload attributes in the XML files of the human tasks involved in this extended workflow are exactly the same.

25.4.5 How to View the Generated Human Task Activity

When you have completed modeling the human task activity, the human task is generated in the designer.

[Figure 25-77](#) shows how a workflow interaction is modeled in Oracle BPEL Process Manager. [Figure 25-77](#) also illustrates the interaction when no BPEL callbacks are modeled. In this case, after a task is complete, the BPEL process is called back with the completed task. No intermediary events are propagated to the BPEL process instance. It is recommended that any user customizations be done in the first assign, `AssignTaskAttributes`, and that `AssignSystemTaskAttributes` not be changed.

Figure 25–77 Workflow Interaction Modeling

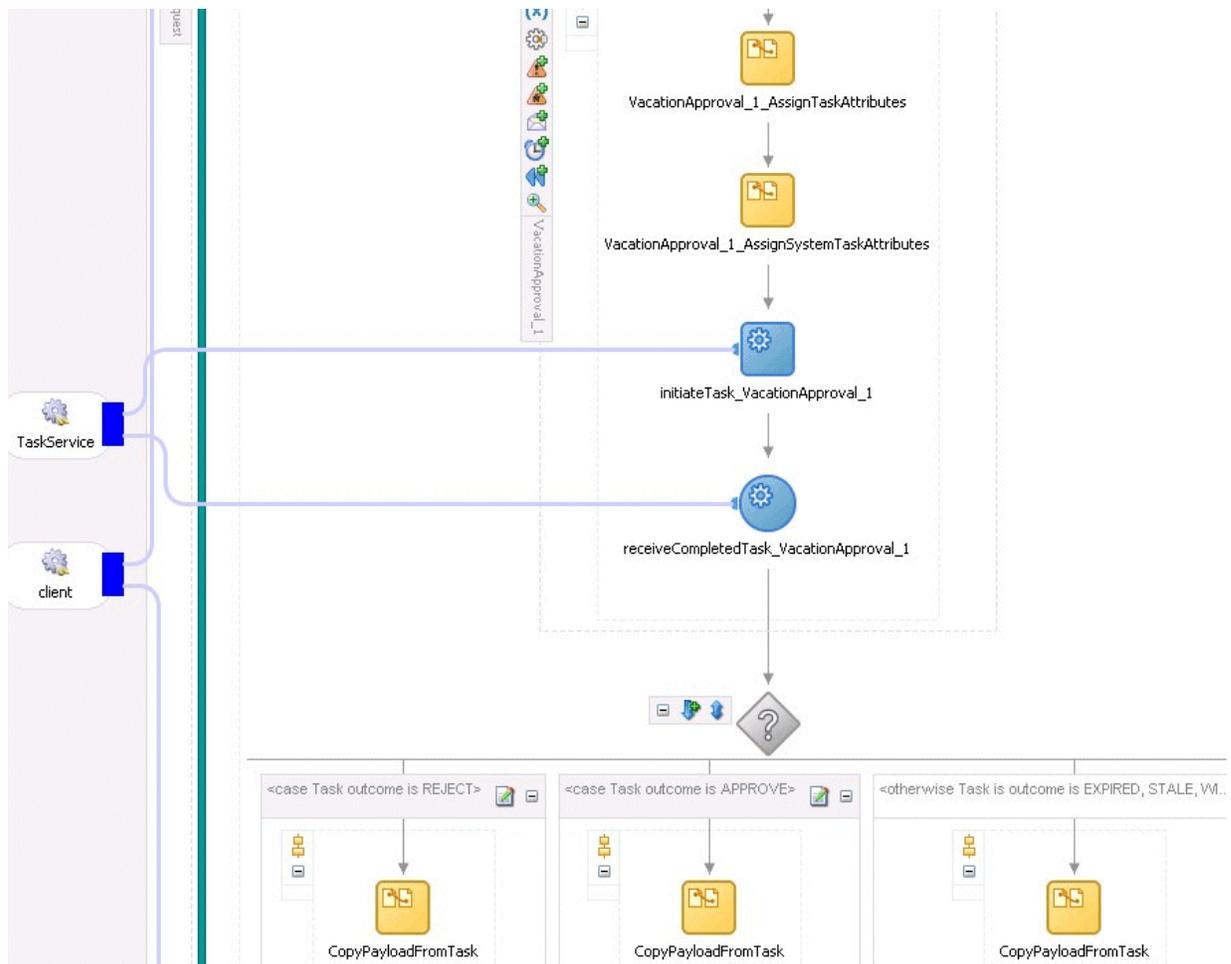


Click the **Expand** icon next to the human task activity in Oracle BPEL Designer to display its contents, as shown in [Figure 25–78](#).

Figure 25–78 Expanding the Human Task Activity



[Figure 25–79](#) shows the workflow interaction in Oracle BPEL Designer.

Figure 25–79 Workflow Interaction Modeling in Oracle JDeveloper


25.4.5.1 Invoking BPEL Callbacks

If intermediary events must be propagated to the BPEL process instance, select the **Allow task and routing customization in BPEL callbacks** checkbox in the **Advanced Settings** section of the Human Task Editor. When these options are selected, the workflow service invokes callbacks in the BPEL instance during each update of the task. The callbacks are listed in the `TaskService.wsdl` file and described as follows:

- `onTaskCompleted`
This callback is invoked when the task is completed, expired, withdrawn, or errored.
- `onTaskAssigned`
This callback is invoked when the task is assigned to a new set of assignees due to the following actions:
 - Outcome update
 - Skip current assignment
 - Override routing slip
- `onTaskUpdated`

This callback is invoked for any other update to the task that does not fall in the `onTaskComplete` or `onTaskAssigned` callback. This includes updates on tasks due to a request for information, a submittal of information, an escalation, a reassign, and so on.

- `onSubTaskUpdated`

This callback is invoked for any update to a subtask.

[Figure 25–80](#) shows how a workflow interaction with callbacks is modeled in Oracle BPEL Process Manager. After this task is initiated, a while loop is used to receive messages until the task is complete. The while loop contains a pick with four `onMessage` branches — one for each of the above-mentioned callback operations. The workflow interaction works fine even if nothing is changed in the `onMessage` branches, meaning that customizations in the `onMessage` branches are not required.

In this scenario, a workflow context is captured in the BPEL instance. This context can be used for all interaction with the workflow services. For example, if you want to reassign a task if it is assigned to a group, then you need the workflow context for the `reassignTask` operation on the task service.

It is recommended that any user customizations be done in the first `assign`, `AssignTaskAttributes`, and that `AssignSystemTaskAttributes` not be changed.

Figure 25–80 Workflow Interaction Modeling (with Callbacks)

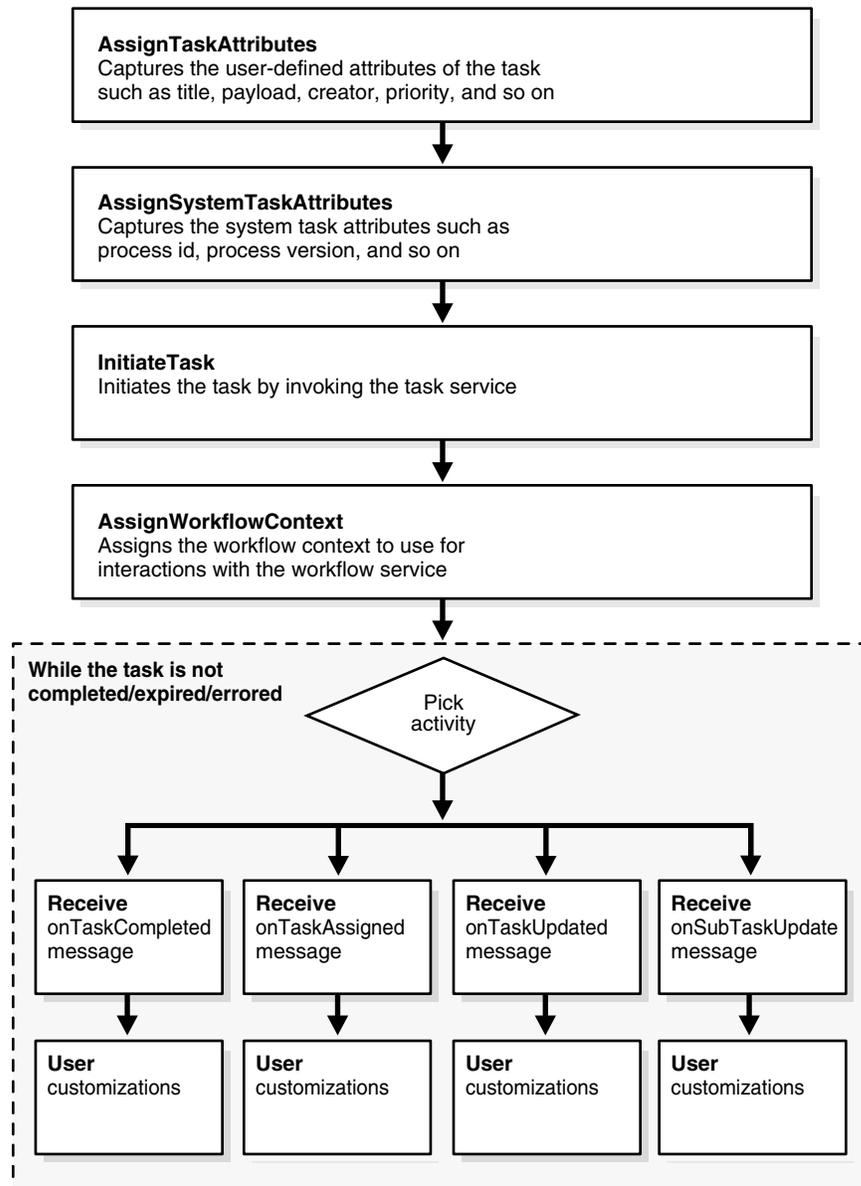
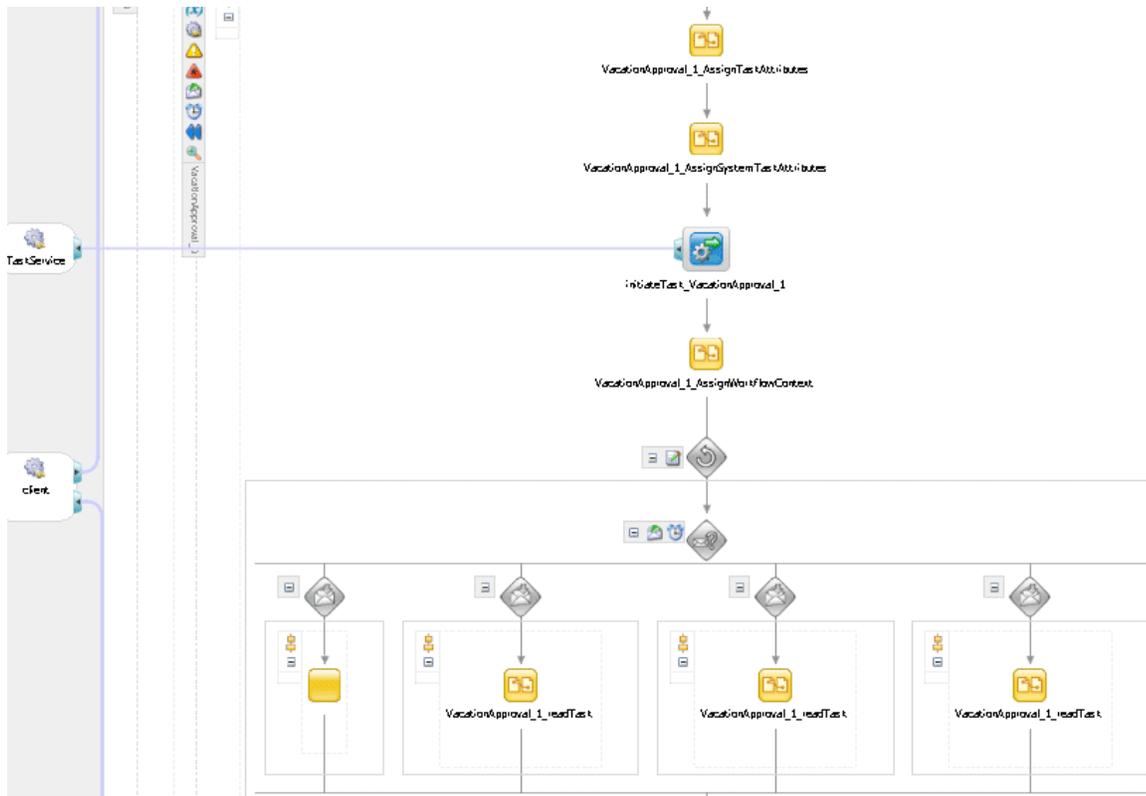


Figure 25–81 shows a workflow interaction in Oracle JDeveloper.

Figure 25–81 Workflow Interaction Modeling (with Callbacks) in Oracle JDeveloper



25.4.6 What You May Need to Know About Changing the Generated Human Task Activity

If you must change a generated human task activity, note the following details:

- Do *not* modify the assign tasks that are automatically created in a switch activity when you add a human task to a BPEL process flow. Instead, add a new assign activity outside the switch activity.
- If the parameter passed into a human task is modified (for example, you change the parameter type in the Edit Task Parameter dialog), you must open the human task activity in the BPEL process flow and click OK to correct the references to the payload variable. Not doing so causes the parameter name to change and become uneditable.

If the task outcomes in the Human Task Editor are modified, you must edit the human task activity and click OK. The switch case is then updated based on the changes to the outcomes.

25.4.7 What You May Need to Know About Deleting a Partner Link Generated by a Human Task

Deleting a partner link that was generated by a human task (for example, *human_task_name.TaskService* in the **Partner Links** swimlane) causes the human task to become unusable. If you delete the partner link, you must delete the human task activity in Oracle BPEL Designer and start over again.

25.4.8 How to Define Outcome-Based Modeling

In many cases, the outcome of a task determines the flow of the business process. To facilitate modeling of the business logic, when a user task is generated, a BPEL switch activity is also generated with prebuilt BPEL case activities. By default, one case branch is created for each outcome selected during creation of the task. An otherwise branch is also generated in the switch to represent cases when the task is withdrawn, expired, or errored.

25.4.8.1 Specifying Payload Updates

The task carries a payload in it. If the payload is set from a business process variable, then an assign activity with the name `copyPayloadFromTask` is created in each of the case and otherwise branches to copy the payload from the task back to its source. If the payload is expressed as other XPath expressions (such as `ora:getNodes(...)`), then this assign is not created because of the lack of a process variable to copy the payload back. If the payload does not require modification, then this assign can be removed.

25.4.8.2 Using Case Statements for Other Task Conclusions

By default, the switch activity contains case statements for the outcomes only. The other task conclusions are captured in the otherwise branch. These conclusions are as follows:

- The task is withdrawn
- The task is errored
- The task is expired

If business logic must be added for each of these other conclusions, then case statements can be added for each of the preceding conditions. The case statements can be created as shown in the following BPEL segment. The XPath conditions for the other conclusions in the case activities for each of the preceding cases are shown in bold in [Example 25–2](#).

Example 25–2 XPath Conditions for Other Conclusions in the Case Activities

```
<switch name="taskSwitch">
  <case condition="bpws:getVariableData('SequentialWorkflowVar1',
'/task:task/task:state') = 'COMPLETED' and
bpws:getVariableData('SequentialWorkflowVar1', '/task:task/task:conclusion') =
'ACCEPT'">
    <bpelx:annotation>
      <bpelx:pattern>Task outcome is ACCEPT
    </bpelx:pattern>
    </bpelx:annotation>
    ...
  </case>
  <case condition=
"bpws:getVariableData('SequentialWorkflowVar1', '/task:task/task:state') =
'WITHDRAWN'">
    <bpelx:annotation>
      <bpelx:pattern>Task is withdrawn
    </bpelx:pattern>
    </bpelx:annotation>
    ...
  </case>
  <case condition=
"bpws:getVariableData('SequentialWorkflowVar1', '/task:task/task:state') =
```

```

    'EXPIRED'">
      <bpelx:annotation>
        <bpelx:pattern>Task is expired
      </bpelx:pattern>
    </bpelx:annotation>
    ...
  </case>
  <case condition=
"bpws:getVariableData('SequentialWorkflowVar1', '/task:task/task:state') =
'ERRORED' ">
    <bpelx:annotation>
      <bpelx:pattern>Task is errored
    </bpelx:pattern>
  </bpelx:annotation>
  ...
</case>
<otherwise>
  <bpelx:annotation>
    <bpelx:pattern>Task is EXPIRED, WITHDRAWN or ERRORED
  </bpelx:pattern>
</bpelx:annotation>
  ...
</otherwise>
</switch>

```

Designing Task Display Forms for Human Tasks

The human workflow service creates tasks for users to interact with the business process. Each task has two parts—the task metadata and the task form. The task form is used to display the contents of the task to the user’s worklist.

Oracle BPM Worklist displays all worklist tasks that are assigned to a user or a group. When a worklist user drills down into a specific task, the task display form renders the details of that task. For example, an expense approval task may show a form with line items for various expenses, and a help desk task form may show details such as severity, problem location, and so on.

This chapter describes how to design and customize task display forms using ADF task flows in Oracle JDeveloper.

This chapter contains the following sections:

- [Section 26.1, "Introduction to the Task Display Form"](#)
- [Section 26.2, "Associating the Task Flow with the Task Service"](#)
- [Section 26.3, "Creating an ADF Task Flow Based on a Human Task"](#)
- [Section 26.4, "Creating a Task Display Form"](#)
- [Section 26.5, "Refreshing Data Controls When the Task XSD Changes"](#)
- [Section 26.6, "Securing the Task Flow Application"](#)
- [Section 26.7, "Creating an Email Notification"](#)
- [Section 26.8, "Deploying a Composite Application with a Task Flow"](#)
- [Section 26.9, "Displaying a Task Display Form in the Worklist"](#)
- [Section 26.10, "Displaying a Task in an Email Notification."](#)

26.1 Introduction to the Task Display Form

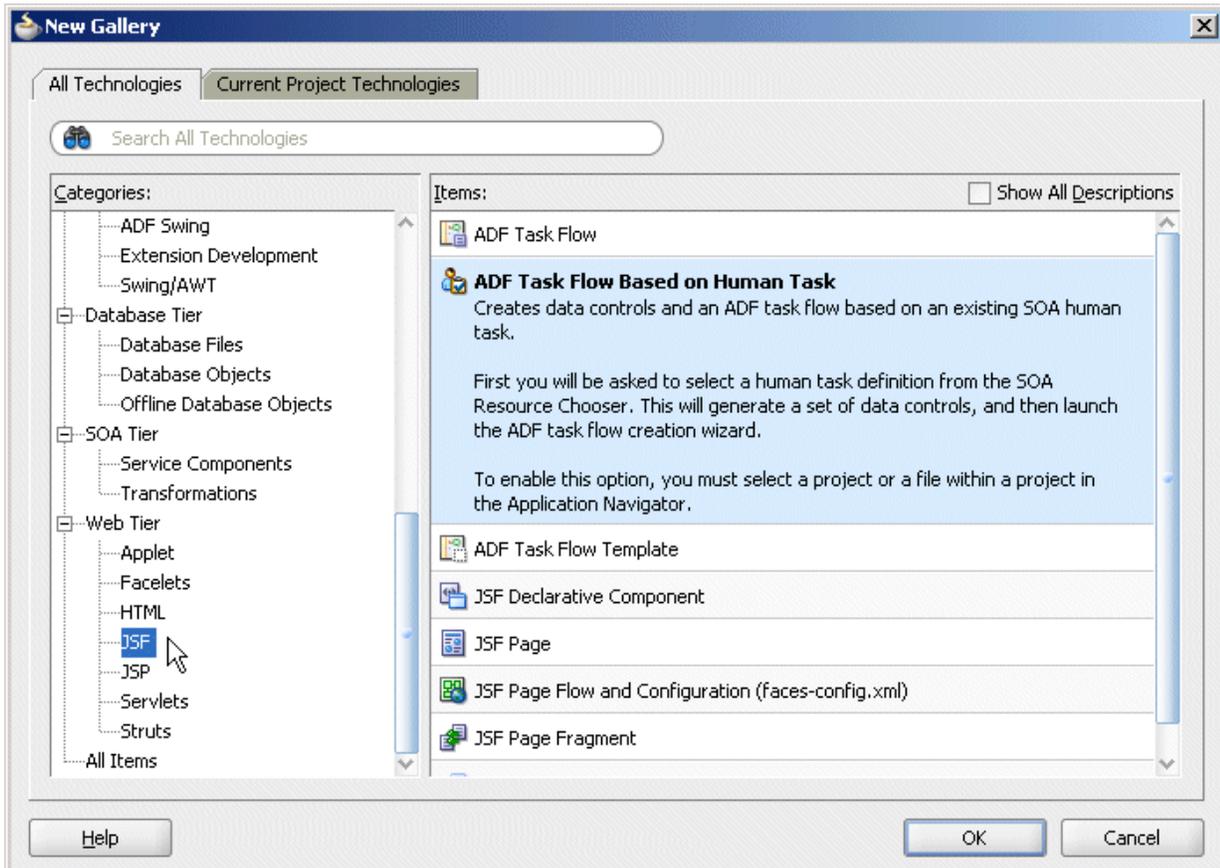
If your SOA composite includes a human task, then you need a way for users to interact with the task. The integrated development environment of Oracle SOA Suite includes Oracle Application Development Framework (Oracle ADF) for this purpose. With Oracle ADF, you can design a task display form that depicts the human task in the SOA composite.

The task display form is a Java Server Page XML (.jspx) file that you create in the Oracle JDeveloper designer where you created the SOA composite containing the human task. Note that you must set the page encoding to UTF-8 in Oracle JDeveloper

before creating the Java Server Page XML file. You can do this in Oracle JDeveloper by choosing **Tools > Preferences > Environment**, and selecting UTF-8 using the **Encoding** dropdown list.

Figure 26–1 shows the Oracle JDeveloper **ADF Task Flow Based on Human Task** option where you start creating a task display form.

Figure 26–1 ADF Task Flow Based on a Human Task, in Oracle JDeveloper



26.2 Associating the Task Flow with the Task Service

When you create an ADF task flow based on a human task, you must select a task metadata file to generate the data control. This data control is used to lay out the content on the page and connect to the workflow service engine at execution time to retrieve task content and act on tasks. See Chapter 14, "Getting Started with ADF Task Flows," in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework* for more information.

The `hwtaskflow.xml` file is used to capture the details on connecting with the service engine. By default, it uses remote EJBs to connect to the workflow server. The SOA server URL and port are automatically determined by using WebLogic Server runtime MBeans. However, you can override these by explicitly specifying the URL and port information here.

Seed a user that has ORMI privileges so that the task details application can connect to the workflow service. You can seed this user by using Oracle Enterprise Manager Fusion Middleware Control.

26.3 Creating an ADF Task Flow Based on a Human Task

ADF task flows are used to model the user interface for the task details page. You can create the task flow in the same application that contains the human task or in a separate application.

You must have previously created a human task (.task file) as part of a SOA composite before you can create a task flow. See [Chapter 25, "Designing Human Tasks"](#) for how to create the .task file.

If the task flow is in the same application as the human task, create a different project for the task flow. If the SOA composite contains multiple human tasks, create a separate project for each ADF task flow associated with each human task. By using an ADF task flow, you create data controls based on the task parameters and outcomes.

To autogenerate an ADF task form, access the human task in the SOA composite application (form and task are in the same application). See [Section 26.3.1, "How To Autogenerate an ADF Task Flow for a Human Task,"](#) for more information.

To create an ADF task form in a separate application, create the new application and project and browse for the .task file for the human task. See [Section 26.3.2, "How To Create an ADF Task Flow Based on a Human Task,"](#) for more information.

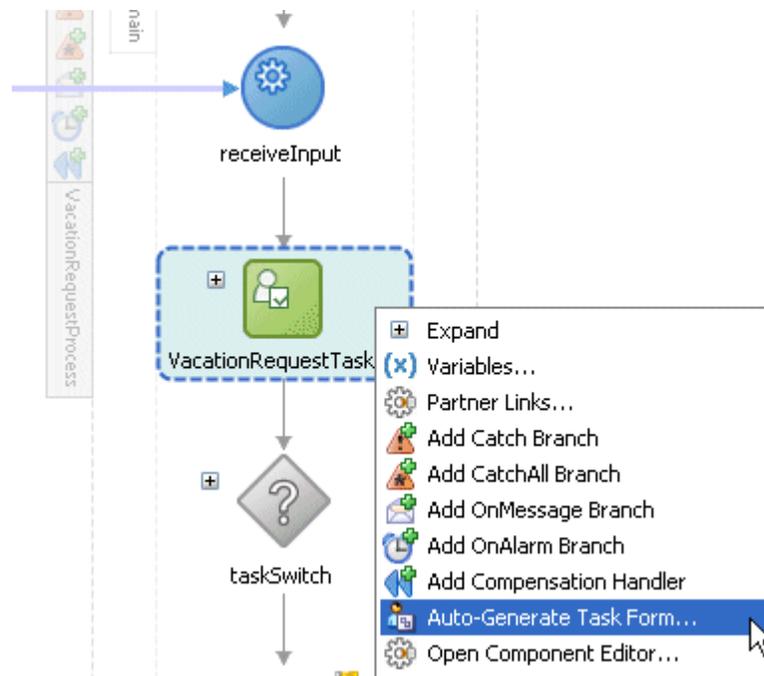
26.3.1 How To Autogenerate an ADF Task Flow for a Human Task

The .task file that specifies the human task is easily associated with the task flow when the two are located in the same application.

To autogenerate an ADF task flow for a human task:

1. Open the BPEL process within the SOA composite application.
2. Right-click the human task activity and select **Auto-Generate Task Form**, as shown in [Figure 26–2](#).

Figure 26–2 Autogenerating a Task Display Form



3. Provide a project name and a directory path (or use the default), and click **OK**.

The form is displayed in the designer, in the `taskDetails1.jspx` tab.

The task flow and task display form are complete and ready to be deployed.

26.3.2 How To Create an ADF Task Flow Based on a Human Task

The **ADF Task Flow Based on Human Task** option (shown in [Figure 26–1](#)) creates an ADF task flow and additional artifacts to make deployment easier. When you select the `.task` file to associate with the ADF task flow, human task data controls are created based on the task parameters and outcomes. These are then available to use in the JSPX page. You must have access to the SOA composite project while creating the task flow project.

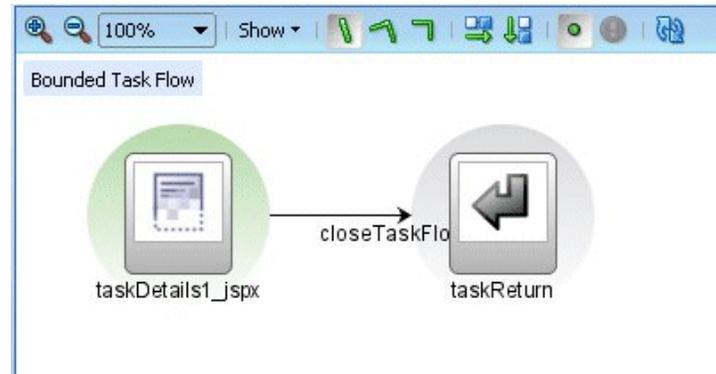
To create an ADF task flow based on a human task:

1. From the **File** main menu, select **New > Applications > SOA Application**.
2. Click **OK**.
3. Provide an application name and directory information (or accept the default), and click **Finish**.
4. Right-click the project name and select **New**.
5. Under **Web Tier**, select **JSF**.
6. Select **ADF Task Flow Based on Human Task** and click **OK**.
7. In the SOA Resource Browser, find and select the `.task` file where you defined the human task and click **OK**.
 - a. If the human task is in the same application as the task definition, then click **File** to use the file browser to navigate to the `.task` file, which is typically in the composite directory.
 - b. If the human task is in a different application, then click **Resource Palette** to use the MDS resource catalog and find the `.task` file in the composite application.
 - c. If the `.task` file is located within the current application, then click **Application**.

This displays the Create Task Flow dialog and creates the data controls.

8. In the Create Task Flow dialog, accept the defaults and click **OK**.

The `taskDetails1.jspx` icon appears in the designer, as shown in [Figure 26–3](#).

Figure 26–3 The taskDetails1_jspX Icon

The task flow has a view, a control flow, and a task return.

To continue creating the task display form, see the following:

- [Section 26.4.1, "How To Create a Task Display Form Using the Complete Task with Payload Drop Handler."](#)

or

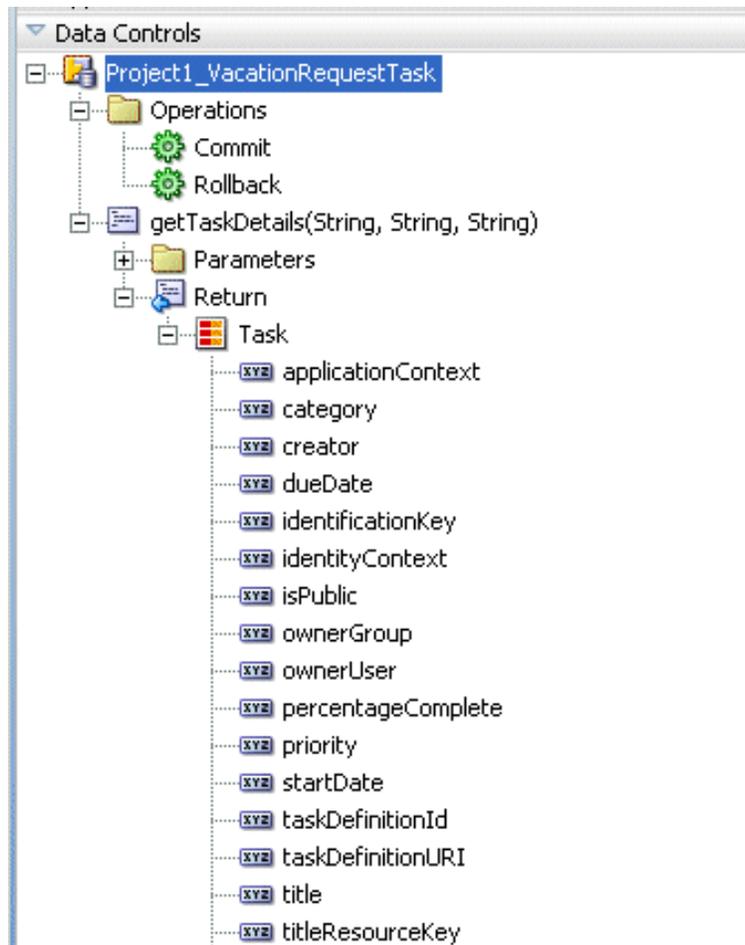
- [Section 26.4.2, "How To Create Task Display Form Regions Using Individual Drop Handlers."](#)

26.3.3 What Happens When You Create an ADF Task Flow Based on a Human Task

With an ADF task flow based on a human task, the task flow application has task data controls that wire the task form with the workflow services. The data controls provide the following:

- Various parameters and operations to access task data and act on it
- Drop handlers with which you can create interface regions to display the contents of the task

The human task-aware data controls appear in the **Data Controls** panel of the Oracle JDeveloper Application Navigator, as shown in [Figure 26–4](#).

Figure 26–4 The Task Collection in the Data Controls Panel

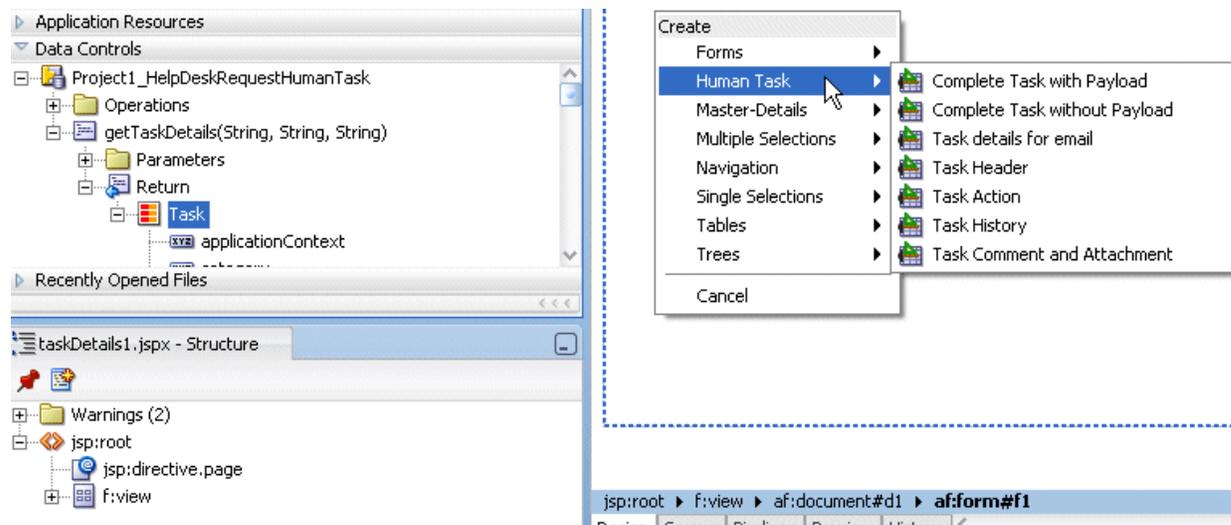
The data controls for the task (represented by the **Task** node in the figure) have drop handlers to render the task display form. See [Section 26.4, "Creating a Task Display Form,"](#) for more information.

26.4 Creating a Task Display Form

Use the human task drop handler to create a task display form. The human task drop handler is designed to autogenerate the following:

- Complete task with payload
- Complete task without payload
- Task details for email
- Task header
- Task action
- Task history
- Task comment and attachment

The human task drop handler appears in the context menu of the designer, as shown in [Figure 26–5](#).

Figure 26–5 Human Task Drop Handler for Creating the Task Display Form

Other ADF drop handlers—for forms, tables, trees, and so on (shown in [Figure 26–5](#))—can also be used to create task display forms. See *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework* for more information about standard ADF drop handlers.

Complete Task with Payload

This option creates the combination of all the preceding task display form components (the task header, task history, task actions, and task comments and attachments), plus the interface for the payload. The payload interface is created as follows:

- All text nodes are created as text input fields.
- If an element has `maxOccurs="unbounded"`, then it appears as a table.
- Nested tables are not rendered; that is, if an element has `maxOccurs="unbounded"` and it has a child with `maxOccurs="unbounded"`, then the child element is not rendered.
- If there are multiple levels of nesting, then drag and drop the individual sections and use a standard ADF drop handler.

Complete Task without Payload

This option creates the combination of all of the preceding task display form components (the task header, task history, task actions, and task comments and attachments).

Task Details for Email

This option creates an ADF region that renders well when sent by email. It generates the form shown in [Figure 26–6](#).

Figure 26–6 Task Display Form for Email Notification

Task Number: <input type="text" value="#{...taskNumber.inputValue}"/>	Creator: <input type="text" value="#{...creator.inputValue}"/>	Assignees: <input type="text" value="#{...id}[#{...}]"/>
State: <input type="text" value="#{...}"/>	Created Date: <input type="text" value="#{...createdDate.inputValue}"/>	Acquired By: <input type="text" value="#{...acquiredBy.inputValue}"/>
Outcome: <input type="text" value="#{...outcome.inputValue}"/>	Updated Date: <input type="text" value="#{...updatedDate.inputValue}"/>	
Priority: <input type="text" value="value"/>	Expiration Date: <input type="text" value="#{...expirationDate.inputValue}"/>	

<input type="text" value="#{...hints.label}"/>	<input type="text" value="#{...Location.inputValue}"/>
<input type="text" value="#{...hints.label}"/>	<input type="text" value="#{...type.inputValue}"/>
<input type="text" value="#{...hints.label}"/>	<input type="text" value="#{...problemDescription.inputValue}"/>
<input type="text" value="#{...hints.label}"/>	<input type="text" value="#{...severity.inputValue}"/>
<input type="text" value="#{...hints.label}"/>	<input type="text" value="#{...status.inputValue}"/>

<input type="text" value="#{...hints.label}"/>	<input type="text" value="#{...ID.inputValue}"/>
<input type="text" value="#{...hints.label}"/>	<input type="text" value="#{...FirstName.inputValue}"/>
<input type="text" value="#{...hints.label}"/>	<input type="text" value="#{...LastName.inputValue}"/>
<input type="text" value="#{...hints.label}"/>	<input type="text" value="#{...Email.inputValue}"/>
<input type="text" value="#{...hints.label}"/>	<input type="text" value="#{...phone.inputValue}"/>

See [Section 26.7, "Creating an Email Notification,"](#) for more information.

Task Header

All the standard header fields are added to the task display form. This includes the task number and title; the state, outcome, and priority of the BPEL process, and information about who created, updated, claimed, or is assigned to the task. The header also displays dates related to task creation, last modification, and expiration. You can add or remove header fields as required for your task display.

[Figure 26–7](#) shows an example of header information.

Figure 26–7 Header Information

Details
toolbar

Assignees <input type="text" value="#{...displayName}"/>	Expiration Date <input type="text" value="#{...expirationDate.inputValue}"/>	Task Number <input type="text" value="#{...taskNumber.inputValue}"/>
Creator <input type="text" value="#{...creator.inputValue}"/>	Acquired By <input type="text" value="#{...acquiredBy.inputValue}"/>	Priority <input type="text" value="value"/>
Created <input type="text" value="#{...createdDate.inputValue}"/>	Due Date <input type="text" value="#{...dueDate.inputValue}"/>	State <input type="text" value="#{...}"/>
Updated <input type="text" value="#{...updatedDate.inputValue}"/>	Outcome <input type="text" value="#{...outcome.inputValue}"/>	

Buttons for task actions are also created in the header, as shown in [Figure 26–8](#).

Figure 26–8 Task Header: Task Action Buttons

Task Actions

The following task actions appear from the **Actions** dropdown list or as buttons. The tasks that appear depend on the state of the task (assigned, completed, and so on) and the privileges that are granted to the user viewing the task. For example, when a task is assigned to a group, only the **Claim** button appears. After the task is claimed, other actions such as **Reject** and **Approve** appear.

The first three custom actions appear on the task form as buttons: **Claim**, **Dismiss**, and **Resume**. Only those buttons applicable to the task appear. Other actions are displayed under the **Actions** list, starting with **Request for Information**, **Reassign**, and **Route**. Systems actions—**Withdraw**, **Pushback**, **Escalate**, **Release**, **Suspend**, and **Renew**—follow the custom actions, followed by the **Save** button. These actions require no further dialog to complete.

- **Claim**—A task that is assigned to a group or multiple users must be claimed first. **Claim** is the only action available in the **Task Action** list for group or multiuser assignments. After a task is claimed, all applicable actions are listed.
- **Dismiss**—This action is used for a task that requires the person acting on the task to acknowledge receipt, but not take any action (like an FYI).
- **Resume**—A task that was halted by a **Suspend** action can be worked on again. See **Suspend**.
- **Request for Information**—You can request more information from the task creator or any of the previous assignees. If reapproval is not required, then the task is assigned to the next approver or the next step in the business process.
- **Reassign**—Managers can reassign a task to reportees. A user with `BPMWorkflowReassign` privileges can reassign a task to anyone. The **Reassign** option also provides a **Delegate** function. A task can be delegated to another user or group. The delegated task appears in both the original user's and the delegated user's worklists. The delegated user can act on behalf of the original assignee, and has the same privileges for that task as the original assignee.
- **Route**—If there is no predetermined sequence of approvers or if the workflow was designed to permit ad hoc routing, then the task can be routed in an ad hoc fashion. For such tasks, a **Route** button appears on the task details page. From the Routing page, you can look up one or more users for routing. When you specify multiple assignees, you can choose whether the list of assignees is for simple (group assignment to all users), sequential, or parallel assignment. In the case of parallel assignment, you provide the percentage of votes required for approval.
- **Withdraw**—Only the task creator can withdraw (cancel) the task. The **Comments** area is available for an optional comment. The business process determines what happens next.
- **Pushback**—This action sends a task up one level in the workflow to the previous assignee.
- **Escalate**—An escalated task is assigned to the user's manager. The **Comments** area is available for an optional comment.

- **Release**—Releasing a task makes it available to other assignees. A task assigned to a group or multiple users can then be claimed by the other assignees.
- **Suspend**—This action suspends the expiration date indefinitely, until the task is resumed. Suspending and resuming tasks are available only to users who have been granted the `BPMWorkflowSuspend` role. Other users can access the task by selecting **Previous** in the task filter or by looking up tasks in the Suspended status. Buttons that update a task are disabled after suspension.
- **Renew**—Renewing a task extends the task expiration date seven days (P7D is the default). The renewal duration is controlled from Oracle Enterprise Manager Grid Control Console. A renewal appears in the task history. The **Comments** area is available for an optional comment.
- **Save**—Changes to the task are saved.

While you are creating a task display form, all possible system action buttons appear, although only those actions that are appropriate for the task state and fit the user’s privileges appear in the worklist.

Task History

The history of task actions appears on the task details page, and is displayed in the worklist as a history table. The history includes the following fields:

- Version number
- Participant name—the person who acted on the task
- Action—for example, if the task was approved or assigned
- Updated By—name of the person who last updated the task
- Action date

See [Figure 27–19, "History: Graphical View"](#) and [Figure 27–20, "History: Full Task Actions"](#) for how task history is displayed in Oracle BPM Worklist, including the options to take a history snapshot, list future participants, and list full task actions.

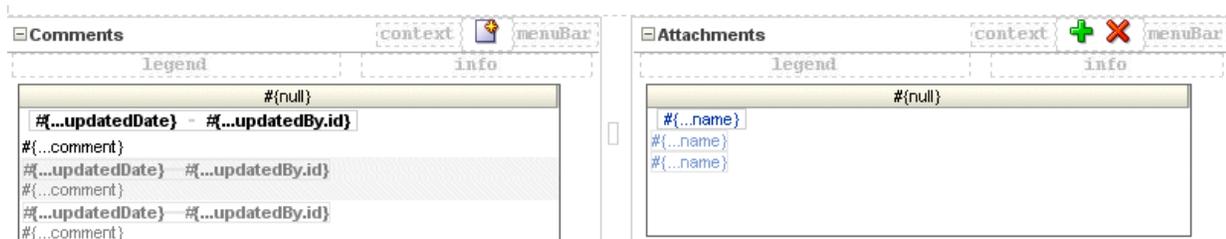
Task Comments and Attachments

A trail of comments with the comment date and comment writer’s user name is maintained throughout the life cycle of a task.

Files or reference URLs associated with a task can be added by any of the human task participants.

[Figure 26–9](#) shows an example of the comments and attachments region.

Figure 26–9 Comments and Attachment Region



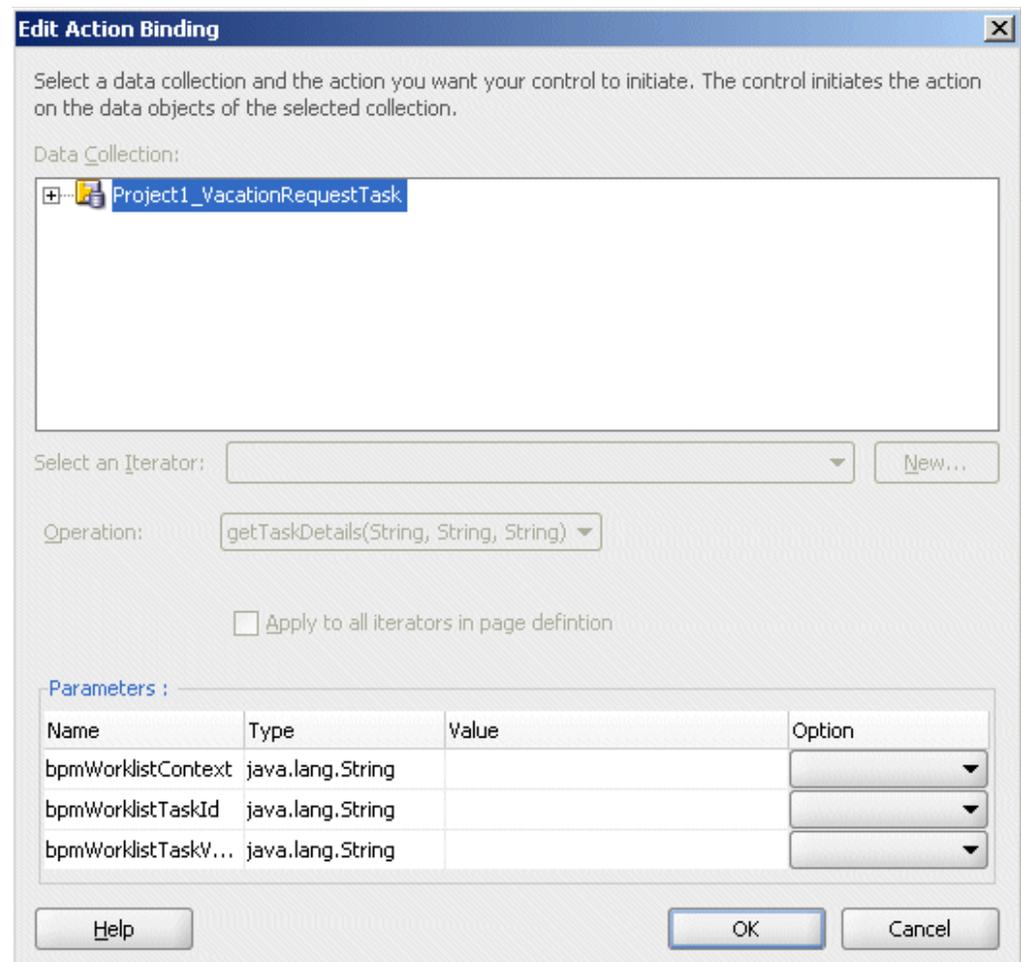
26.4.1 How To Create a Task Display Form Using the Complete Task with Payload Drop Handler

The following steps describe how to use a drop handler that creates the task display form, including the payload, without building each region individually. To build each region individually, see [Section 26.4.2, "How To Create Task Display Form Regions Using Individual Drop Handlers."](#)

To create a task display form using the Complete Task with Payload drop handler:

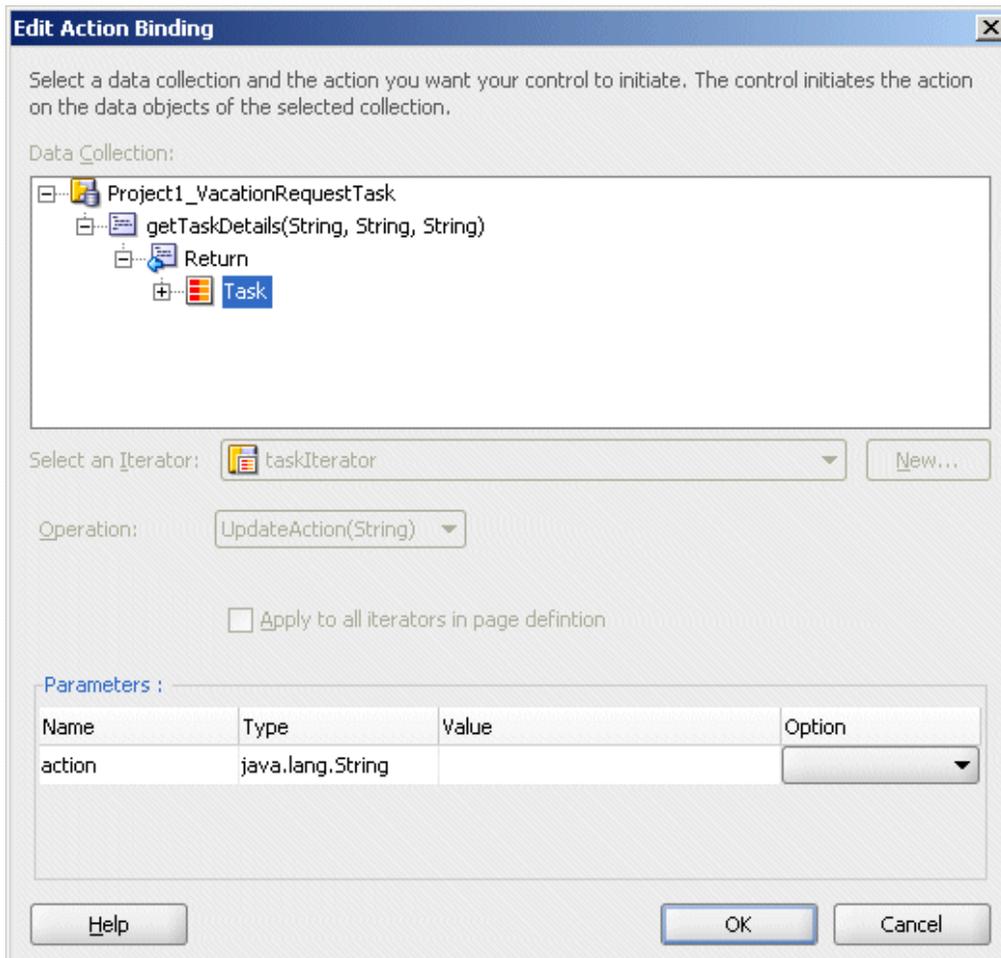
1. In the designer, double-click `taskDetails1.jspx`.
2. In the Create JSF Page dialog, provide a file name and directory information (or accept the defaults) and click **OK**.
3. In the **Data Controls** panel of the Application Navigator, expand the human task node, then the `getTaskDetails` node, and then the **Return** node.
4. Drag the **Task** icon into the `taskDetails.jspx` window.
5. Select **Human Task**, and then **Complete Task with Payload**.
6. In the Edit Action Binding dialog, shown in [Figure 26–10](#), click **OK**.

Figure 26–10 Edit the Action Binding



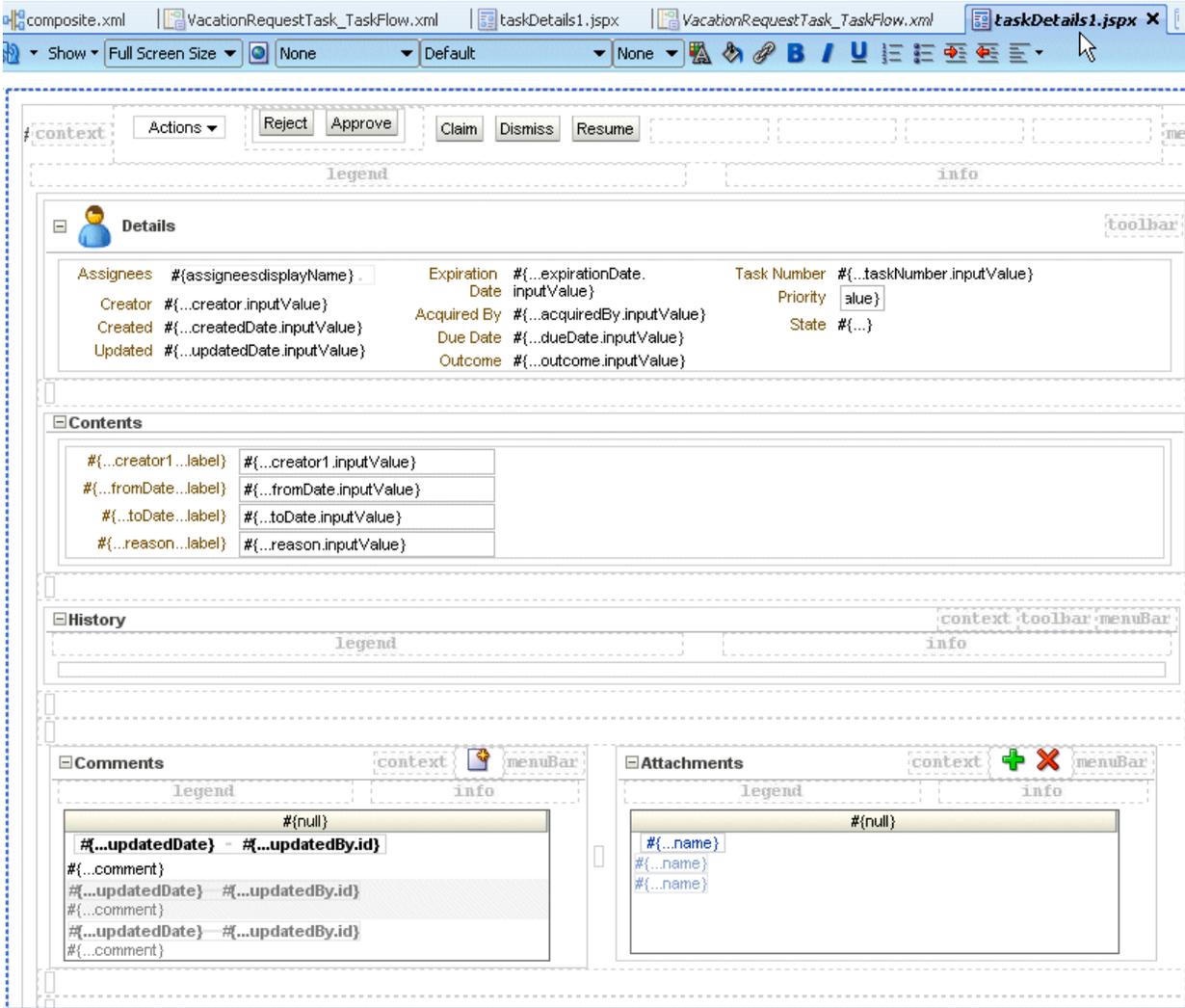
- In the next Edit Action Binding dialog, the data collection is selected, as shown in [Figure 26–11](#); click OK.

Figure 26–11 *Select the Data Collection and Action*



The task display form is displayed, as shown in [Figure 26–12](#).

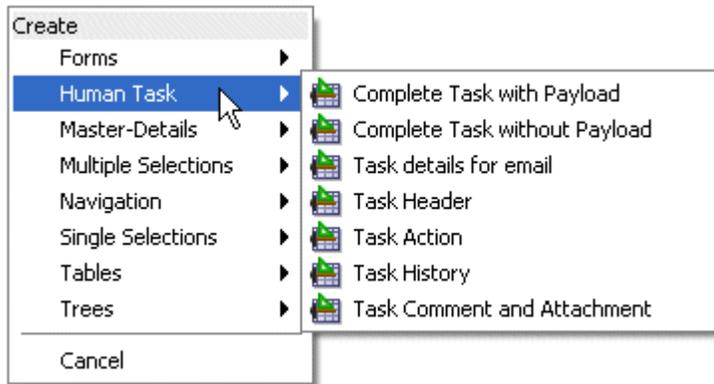
Figure 26–12 Task Display Form



26.4.2 How To Create Task Display Form Regions Using Individual Drop Handlers

You can create a display form with multiple regions using the individual **Task Header**, **Task Action**, **Task History**, and **Task Comment and Attachment** drop handlers shown in [Figure 26–13](#).

Figure 26–13 Using Human Task Drop Handlers



Task Header provides both header and task actions, so you do not need the **Task Action** drop handler when you use **Task Header**. Use **Task Action** when you want the actions dropdown menu and buttons, but not header details.

To create the task display form without building each region individually, see [Section 26.4.1, "How To Create a Task Display Form Using the Complete Task with Payload Drop Handler."](#)

Before you create this task display form, you must have created the following:

- A new application and SOA project, and a human task service.
- An ADF task flow based on the human task. See [Section 24.3.2.2, "How to Create the Vacation Request Process,"](#) for more information.

To create task display form regions using individual drop handlers:

1. In the designer, double-click `taskDetails1.jspx`.
2. In the Create JSF Page dialog, provide a file name and directory information (or accept the defaults) and click **OK**.
3. In the **Data Controls** panel of the Application Navigator, expand the human task node, then the `getTaskDetails` node, and then the **Return** node.
4. Drag the **Task** icon into the `taskDetails.jspx` window.
5. Select **Human Task**, and then **Task Header**.

This creates the **Actions** dropdown list and buttons for task actions, as shown in [Figure 26–14](#), and header details, as shown in [Figure 26–15](#).

Figure 26–14 Designing the Task Display Form: Buttons for Task Actions

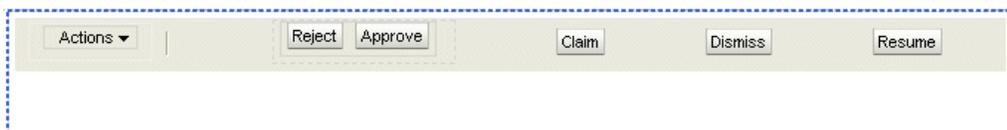


Figure 26–15 Designing the Task Display Form: Task Headers

The screenshot shows a form titled "Details" with a toolbar in the top right corner. The form contains a grid of data fields, each with a JSPX expression for its value:

Assignees	<code>#{...displayName}</code>	Expiration Date	<code>#{...expirationDate.inputValue}</code>	Task Number	<code>#{...taskNumber.inputValue}</code>
Creator	<code>#{...creator.inputValue}</code>	Acquired By	<code>#{...acquiredBy.inputValue}</code>	Priority	<code>#{...priority.inputValue}</code>
Created	<code>#{...createdDate.inputValue}</code>	Due Date	<code>#{...dueDate.inputValue}</code>	State	<code>#{...}</code>
Updated	<code>#{...updatedDate.inputValue}</code>	Outcome	<code>#{...outcome.inputValue}</code>		

6. Drag additional **Task** icons into the JSPX designer, selecting these options with each iteration:

- **Human Task**, then **Task History**
- **Human Task**, then **Task Comment and Attachment**

The task display form now has multiple regions for task action dropdown lists and buttons, task header details, task history, and comments and attachments.

To continue creating the task display form, see Step 1 in [Section 26.4.3, "How To Add the Payload to the Task Display Form."](#)

26.4.3 How To Add the Payload to the Task Display Form

In addition to adding the payload, you can create task display form regions. See Step 1 in [Section 26.4.2, "How To Create Task Display Form Regions Using Individual Drop Handlers."](#)

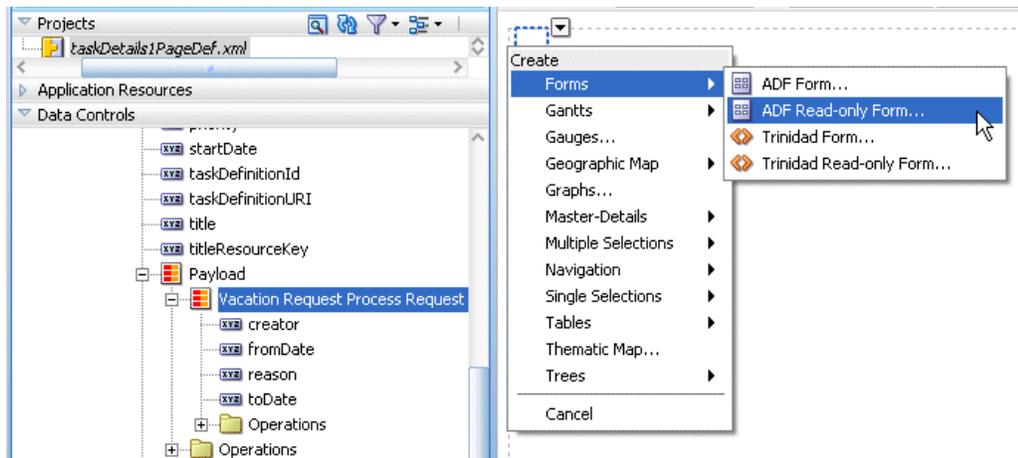
To add the payload to the task display form:

1. From the **Component Palette**, select **ADF Faces**.
2. Expand **Layout**.
3. Drag **Panel Group Layout** between the **Header** and **Comment** sections.
4. In the **Data Controls** panel, expand **Task**, and then **Payload**.
5. Drag the payload data collection to the left of the **Panel Group Layout** area.

An alternative to dropping the payload node onto the form is to expand the payload node and drop the necessary child elements onto the form. For example, to create a read-only form for the `VacationRequest` payload, expand the payload node, drag the `Vacation Request Process Request` node onto the form, and select **Forms > ADF Read-only Form**.

6. From the context menu, select **Forms**, then **ADF Read-only Form**, as shown in [Figure 26–16](#).

Figure 26–16 Adding ADF Read-Only Fields to the Task Display Form Payload Region



7. In the Edit Form Fields dialog, accept the defaults and click **OK**.

This creates read-only fields in the payload region, between the **Details** and **History** sections.

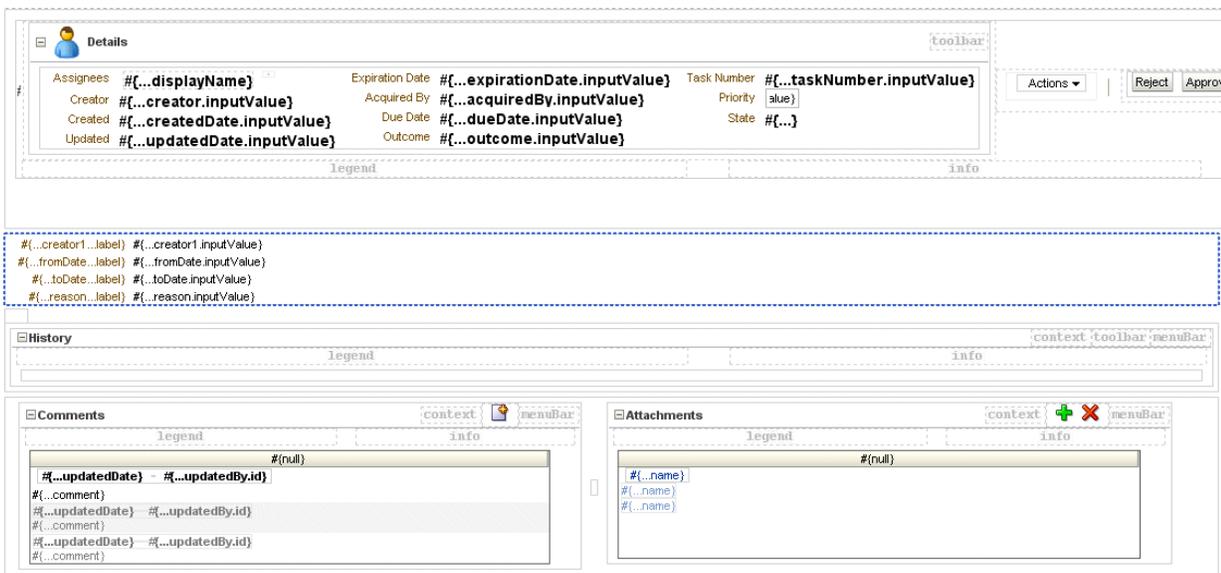
The payload regions appear, as shown in [Figure 26–17](#).

Figure 26–17 The Payload Region of the Task Display Form

```
#{...creator1...label} #{...creator1.inputValue}
#{...fromDate...label} #{...fromDate.inputValue}
#{...toDate...label} #{...toDate.inputValue}
#{...reason...label} #{...reason.inputValue}
```

The task display form, shown in [Figure 26–18](#), is complete and ready to be deployed.

Figure 26–18 The Task Display Form (taskDetails.jspx)



26.4.4 What Happens When You Create a Task Display Form

The form you designed is saved in the `taskDetails.jspx` file at

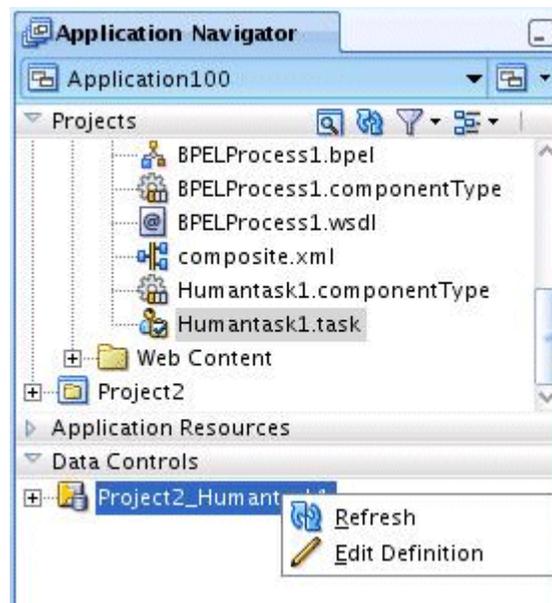
`JDev_Oracle_Home\mywork\task_form_application_name\project_name\public_html`

The task display form is ready to be deployed. See [Section 26.8, "Deploying a Composite Application with a Task Flow."](#)

26.5 Refreshing Data Controls When the Task XSD Changes

When task metadata changes, refresh the *Data Controls view* (XSD changes are not refreshed) that is based on that task metadata. The refresh functionality re-creates the data control. [Figure 26-19](#) shows the **Refresh** option.

Figure 26-19 Refreshing Data Controls



To refresh the data control:

1. Right-click the data control.
2. Select the **Edit Definition** option to display the Refresh Data Control dialog, as shown in [Figure 26-20](#).

Figure 26–20 The Refresh Data Control Button



26.6 Securing the Task Flow Application

You can use any container-based security for securing the task flow. See [Section 29.6.2.1.2, "Requirements for Client Applications For Identity Propagation,"](#) for more information. Form-based authentication and SSO-based authentication are available for web security.

If you are sending a notification as email, do not secure the URL with "/notification/secure" to use container-based security because this is accessed by SOA APIs using an internal context that cannot be created outside of SOA. The URL pattern inside security cannot contain "/" (all URLs) and "//notification".

No additional steps are required for identity propagation. Identity is automatically propagated to the server EJBs.

26.7 Creating an Email Notification

A task display form is used to provide an email notification, if email notification is defined as part of the human task. Options for email notification include:

- Default email notification—Use the first page of the task display form that you create for the human task. The content is sent as an HTML-based email. Images in the task flow are included as attachments so that the notification can be viewed in disconnected mode. However, if you use the **Complete Task with Payload** or **Complete Task without Payload** drop handlers, then the generated JSPX file is not suitable for e-mails. In this case, using the custom email notification is recommended.
- Custom email notification—Use the **Task display for email** drop handler to create an email notification task page.

[Section 29.2, "Notifications from Human Workflow"](#) to review notification settings as part of a human task definition (.task file).

26.7.1 How To Create an Email Notification

To send a custom email notification whose content and layout you have specified (instead of sending the default page), create another JSPX file in which you design an email notification page. Create the notification page by using the custom and standard drop handlers, or use the email notification drop handler. In addition, do the following:

- Add a router to the task flow. The router directs the task flow to send either the email notification page or the default page, depending on the control flow based on the `bpmClientType` page flow scope value.
- Edit the generated inline CSS to customize the page. No additional CSS is included at runtime and the ADF CSS is not available at runtime. See the samples `notification-101` and `notification-102` at http://www.oracle.com/technology/sample_code/products/hwf
- Reference images directly from the HTML or JSF page. (Indirect references, for example, an included JSF that in turn includes the image, are not allowed.)

26.7.1.1 Creating a Task Flow with a Router

The control flow case with a router enables you direct the request to a specific page based on certain parameters. For an ADF task flow based on a human task, you need a special page for email notifications. This section describes how to create a special page for email notifications.

To create a task flow with a router:

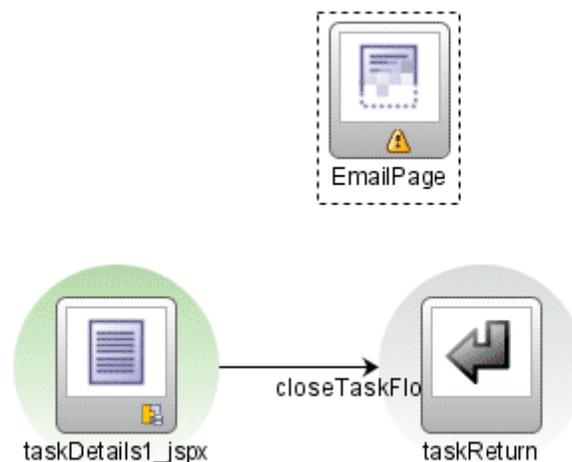
1. In the Application Navigator, expand the task flow project and double-click `project_name_TaskFlow.xml`.

The XML file opens in the designer. In the diagram view, you see the `taskDetails1.jspx` icon.

2. From the **Component Palette**, select **ADF Task Flow**, and drag the **View** icon into the designer.
3. Click `view1` below the icon and enter a name for the email notification page.

[Figure 26–21](#) shows an example using the name **EmailPage**.

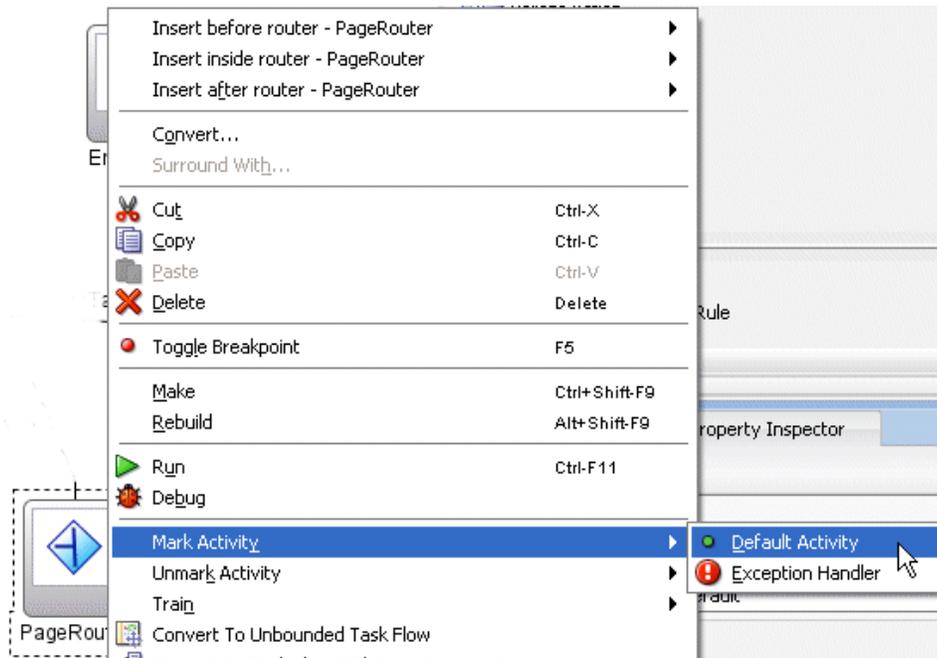
Figure 26–21 Creating the Email Page



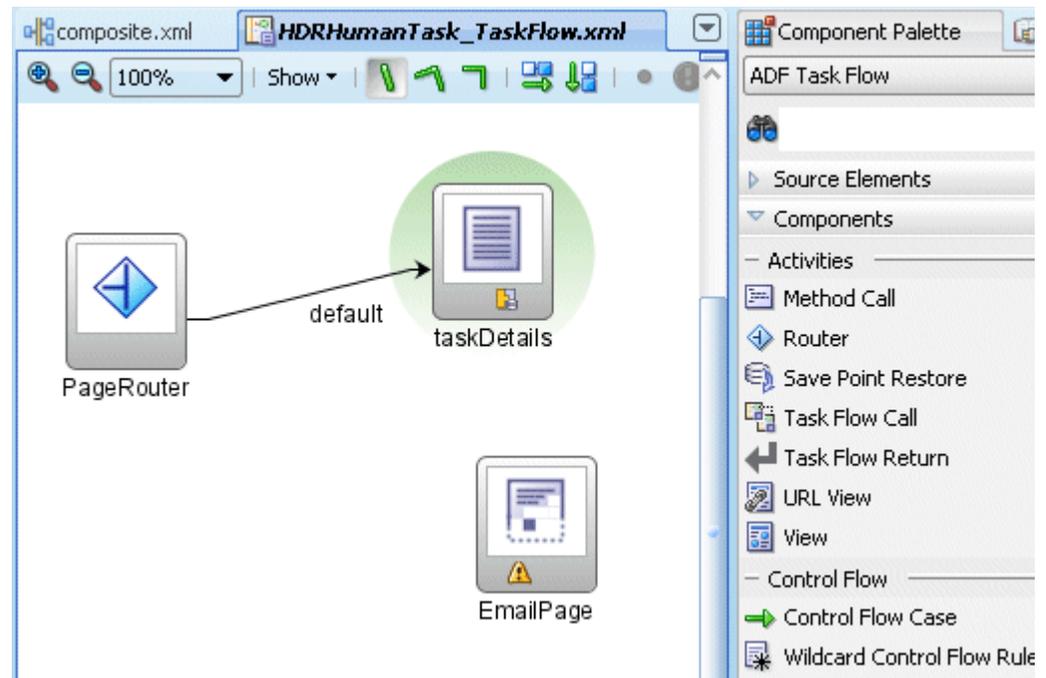
4. From the **Component Palette**, drag **Router** into the designer.

5. Click **router1** below the icon and enter a router name.
 Figure 26–23 shows an example using the name **PageRouter**.
6. To ensure that the router is called, right-click the router icon and click **Mark Activity > Default Activity**, as shown in Figure 26–22.

Figure 26–22 Marking the Router as the Default Activity



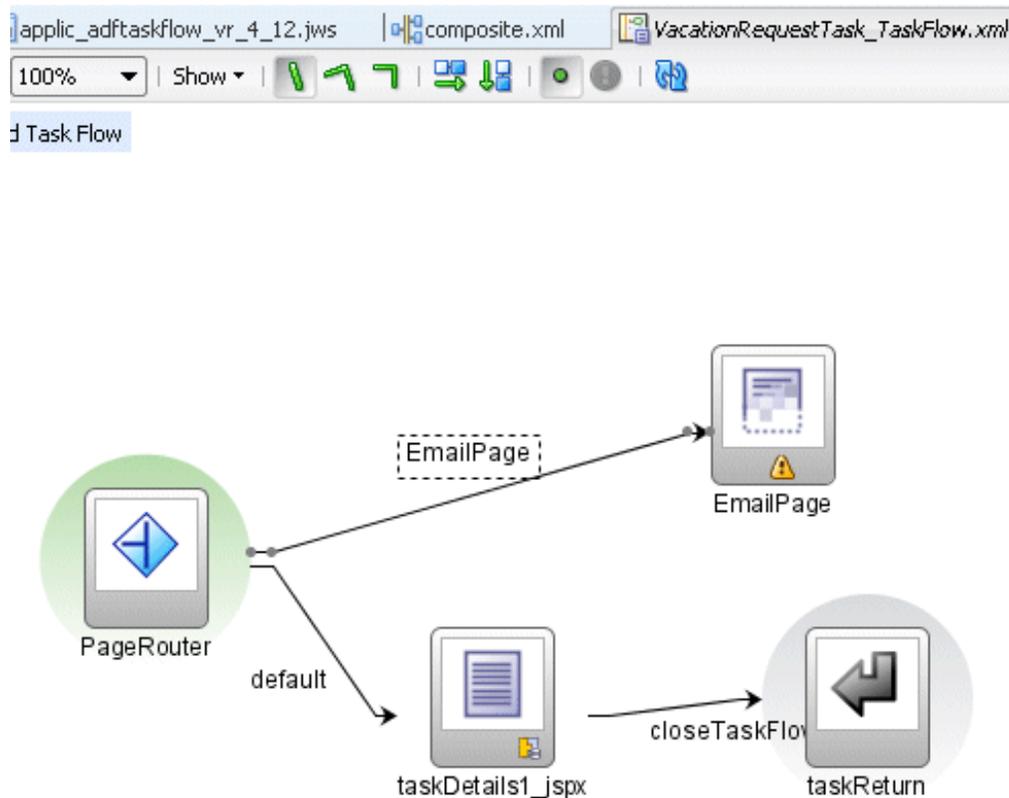
7. Click the **router - router_name - Property Inspector** tab.
8. In the **default-outcome** field, enter `default`.
9. Click **Add**, and in the **Outcome** field, enter the name of the email notification page.
10. Use the Expression Builder to enter the following in the **expression** field:
`#{pageFlowScope.bpmClientType=="notificationClient"}`
11. In the **Component Palette**, click **Control Flow Case**.
12. In the designer, drag from the router page icon to **taskDetails1.jspx**.
 The control flow is automatically labeled **default**, as shown in Figure 26–23.

Figure 26–23 Connecting the Control Flow

13. In the **Component Palette**, click **Control Flow Case**.
14. In the designer, drag from the router page icon to the email notification page icon.
15. Click the **control-flow-case - email_page_name - Property Inspector** tab.
16. From the **from-outcome** list, select the name of the email notification page.

Figure 26–24 shows the completed control flow.

Figure 26–24 Completed Control Flow for an Email Notification



To continue creating the email notification page, see Step 1 in [Section 26.7.1.2, "Creating an Email Notification Page."](#)

26.7.1.2 Creating an Email Notification Page

Creating an email notification page is similar to creating a task display form, with the addition of defining layout and inline styles. See *Oracle Fusion Middleware Web User Interface Developer's Guide for Oracle Application Development Framework* for design information.

To create an email notification page:

1. In the designer, double-click **EmailPage**.
2. In the Create JSF Page dialog, provide a file name and directory information (or accept the defaults) and click **OK**.

The **EmailPage.jspx** tab opens in the designer.

3. From the **Component Palette**, drag any of the **Common Components** (for an image, for example) or **Layout** components into the designer.
4. For the layout component you selected, provide alignment and other details in the **Property Inspector** tab.

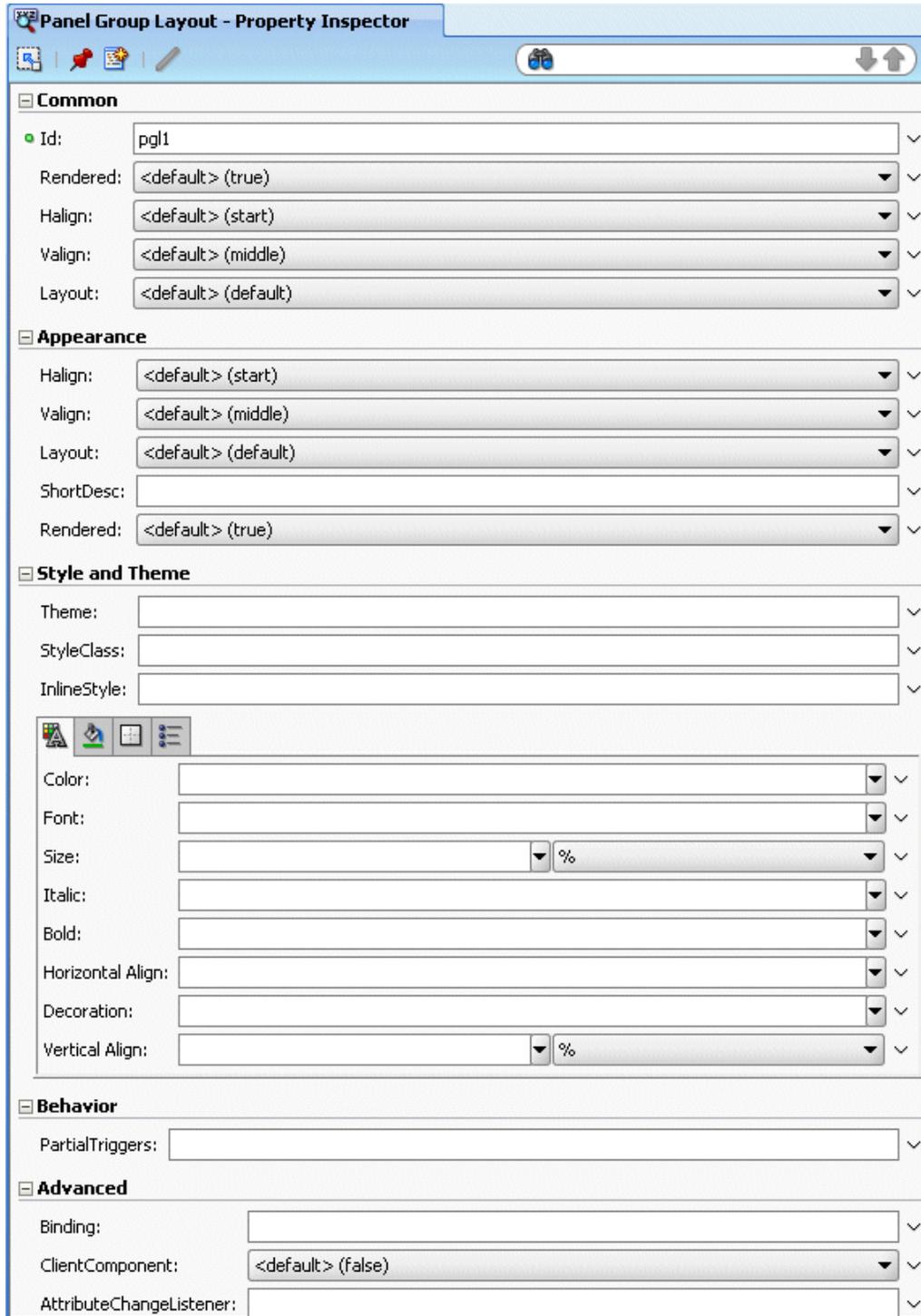
[Figure 26–25](#) shows the layout fields available when **Panel Group Layout** is selected.

Figure 26–25 Specifying a Layout

See *Oracle Fusion Middleware Web User Interface Developer's Guide for Oracle Application Development Framework* for more information about panel group layout.

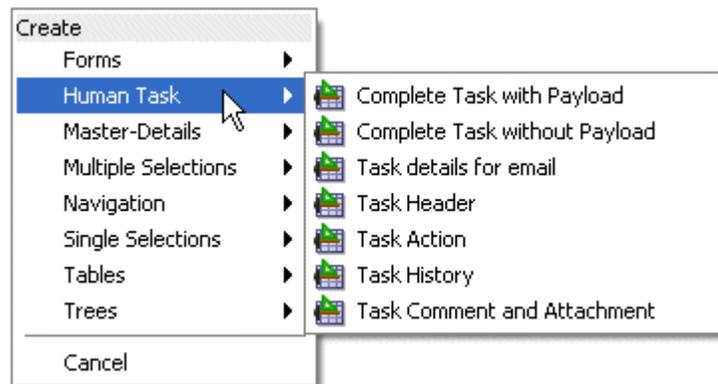
5. Expand **Appearance**, **Style and Theme**, **Behavior**, **Advanced**, **Customization**, and **Annotations** to specify other details, as shown in [Figure 26–26](#).

Figure 26–26 Specifying a Layout: More Details



See Section 2.4.6, "How to Set Component Attributes," in *Oracle Fusion Middleware Web User Interface Developer's Guide for Oracle Application Development Framework*

6. From the **Data Controls** panel, expand the human task node, then the **getTaskDetails** node, and then the **Return** node.
7. Drag **Task** into the panel group layout area.
8. Select **Human Task**, and then **Task details for email**, as shown in [Figure 26–27](#).

Figure 26–27 Human Task Drop Handlers

This drop handler includes a header with inline style, a payload using ADF, and a comment using inline style. Because the payload is dynamically generated, it does not include an inline style.

In general, you can find the inline styles for the Header section for each component and use the same style for the Content section for the respective components.

9. In the Edit Action Bindings dialog, select the data collection and click **OK**.

The email task form is complete and ready to be deployed.

26.7.2 What Happens When You Create an Email Notification Page

The email notification page is sent as HTML content in the email message body. Images on the page are inlined as attachments. Relative URLs are converted to absolute URLs.

26.7.3 What You May Need to Know About Creating an Email Notification Page

A notification may not display correctly in email if the styles used in the fields of the form are not valid for email. Editing the generated inline CSS to customize the page may be required. See [Section 26.7.1, "How To Create an Email Notification,"](#) for more information.

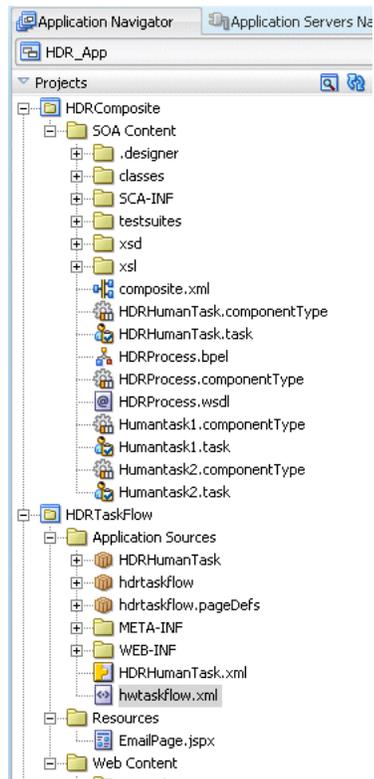
Security issues can also prevent the form from being rendered correctly. See [Section 26.6, "Securing the Task Flow Application,"](#) for more information.

26.8 Deploying a Composite Application with a Task Flow

The composite application containing the task flow must be deployed before you can use the task display form in the Worklist Application. The process for deploying an application with a task flow is basically the same as deploying any SOA composite application, as described in [Section 26.8.2, "How To Deploy a Composite Application with a Task Flow."](#) See [Chapter 43, "Deploying SOA Composite Applications"](#) for more information.

26.8.1 Before Deploying the Task Display Form: Port Changes

If you are not using the default values for RMI or HTTP ports, open the `hwtaskflow.xml` file in Oracle JDeveloper to change values. [Figure 26–28](#) shows the file in the Application Navigator.

Figure 26–28 The *hwtaskflow.xml* File

Example 26–1 shows a sample *hwtaskflow.xml* file with comments on which values can and cannot be changed.

Example 26–1 *Sample hwtaskflow.xml File*

```
<!--Sample hwtaskflow.xml file. This is required for successful deployment of an
ADF Task Flow Based on Human Task application. -->
```

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<hwTaskFlows xmlns="http://xmlns.oracle.com/bpel/workflow/hwTaskFlowProperties">
```

```
    <!-- Name of the client application used to view the tasks, defaults to
'worklist' -->
```

```
    <ApplicationName>worklist</ApplicationName>
```

```
    <!-- Type of ejb lookup used. If not specified, remote lookup is used. Values
- LOCAL, REMOTE, SOAP -->
```

```
    <LookupType>LOCAL</LookupType>
```

```
    <!-- Do not modify this element. Value must be 'false' for deployment to
complete successfully -->
```

```
    <TaskFlowDeploy>>false</TaskFlowDeploy>
```

```
    <!-- Connection details for soa server for remote ejb lookup.
If not specified, default values for ejbProviderUrl is http://localhost/soa-infra
, aliasKeyName is BPM_SERVICES, keyName is BPM_SERVICES -->
```

```

<SoaServer>
  <ejbProviderUrl/>
  <aliasKeyName/>
  <keyName/>
  <connectionName/>
</SoaServer>

<!-- Connection details for server on which task flow is deployed.
If not specified, default values for hostname is localhost,
      httpPort is 8888 and httpsPort is 443 --> -->

<TaskFlowServer>
  <hostName/>
  <httpPort/>
  <httpsPort/>
</TaskFlowServer>

<!-- Task Flow specific properties -->

<hwTaskFlow>
  <WorkflowName></WorkflowName>
  <TaskDefinitionNamespace></TaskDefinitionNamespace>
  <TaskFlowId></TaskFlowId>
  <TaskFlowFileName></TaskFlowFileName>
</hwTaskFlow>
</hwTaskFlows>

```

26.8.2 How To Deploy a Composite Application with a Task Flow

An application server connection is required to do the following.

To deploy a composite application with a task flow:

1. Right-click the composite application name, select **Deploy**, and then *application_name* > **to** > *application_server_connection*.

If you do not have a connection, select **New Connection** and use the Application Server Connection wizard.

2. In the **Select Deployment Targets** dialog, select a server instance.
3. Click **OK**.

26.8.3 How To Redeploy the Task Display Form

If you change the task display form and want to redeploy it, repeat the deployment step. (Right-click the task form application name, select **Deploy**, and then *application_name* > **to** > *application_server_connection*.) A message asking you if you want to undeploy the form is displayed. Click **OK** and deploy the task form again.

26.8.4 How To Deploy a Task Flow as a Separate Application

If you want to deploy the task flow as a separate application, outside of the SOA composite application, then create a new application and project as a container for the task flow. After you deploy the SOA composite application, deploy the task flow application.

26.8.5 How To Deploy a Task Display Form to a non-SOA Oracle WebLogic Server

Follow the steps in these sections to deploy a task display form to a non-SOA Oracle WebLogic Server:

- [Section 26.8.5.1, "Deploying oracle.soa.workflow.jar to a non-SOA Oracle WebLogic Server"](#)
- [Section 26.8.5.2, "Defining the Foreign JNDI Provider on a non-SOA Oracle WebLogic Server"](#)
- [Section 26.8.5.3, "Defining the Foreign JNDI Provider Links on a non-SOA Oracle WebLogic Server"](#)
- [Section 26.8.5.4, "Including a Grant for bpm-services.jar"](#)
- [Section 26.8.5.5, "Deploying the Application"](#)

26.8.5.1 Deploying oracle.soa.workflow.jar to a non-SOA Oracle WebLogic Server

The oracle.soa.workflow.jar shared library is needed on the non-SOA Oracle WebLogic Server. It is available from

```
ORACLE_JDEV_HOME\jdeveloper\soa\modules\oracle.soa.workflow_11.1.1
```

Use Oracle WebLogic Server Administration Console to deploy the JAR file.

To deploy oracle.soa.workflow.jar:

1. Go to Oracle WebLogic Server Administration Console at
`http://remote_hostname:remote_portnumber/console`
2. In the **Domain Structure** area, click **Deployments**.
3. Click **Install**, as shown in [Figure 26–29](#).

Figure 26–29 Oracle WebLogic Server Administration Console: List of Deployments

The screenshot shows the Oracle WebLogic Server Administration Console interface. The top navigation bar includes 'Home', 'Log Out', 'Preferences', 'Record', and 'Help'. The breadcrumb trail indicates the current location is 'Home > Summary of Deployments'. The left sidebar contains several panels: 'Change Center' with a 'View changes and restarts' link, 'Domain Structure' showing a tree view with 'Deployments' selected, 'How do I...' with a list of tasks, and 'System Status' showing 'Health of Running Servers' with 'Failed (0)' and 'Critical (0)' counts.

The main content area is titled 'Summary of Deployments' and has two tabs: 'Control' (selected) and 'Monitoring'. Below the tabs, there is a text block explaining that the page displays a list of Java EE applications and stand-alone application modules. Below this, there is a 'Customize this table' link and a 'Deployments' section. The 'Deployments' section includes a row of buttons: 'Install', 'Update', 'Delete', 'Start', and 'Stop'. Below these buttons is a table with the following data:

<input type="checkbox"/>	Name	State
<input type="checkbox"/>	adf.oracle.domain(1.0,11.1.1.1.0)	Active
<input type="checkbox"/>	adf.oracle.domain.webapp(1.0,11.1.1.1.0)	Active
<input type="checkbox"/>	AqAdapter	Active
<input type="checkbox"/>	b2bui	Installed
<input type="checkbox"/>	DbAdapter	Active
<input type="checkbox"/>	DefaultToDoTaskFlow	Active
<input type="checkbox"/>	DMS Application (11.1.1.1.0)	Active
<input type="checkbox"/>	DocumentReviewTaskFlow	Active
<input type="checkbox"/>	FileAdapter	Active
<input type="checkbox"/>	FtpAdapter	Active

At the bottom of the table, there is another row of buttons: 'Install', 'Update', 'Delete', 'Start', and 'Stop'.

4. In the **Path** field, provide the following path and click **Next**.

```
ORACLE_JDEV_HOME/jdeveloper/soa/modules/oracle.soa.workflow_11.1.1/oracle.soa.workflow.jar
```

5. Keep the same name for the deployment and click **Next**, as shown in Figure 26–30.

Figure 26–30 Oracle WebLogic Server Administration Console: Install Applications Assistant

Home > Summary of Deployments

Install Application Assistant

Back Next Finish Cancel

Optional Settings

You can modify these settings or accept the defaults

General

What do you want to name this deployment?

Name:

Security

What security model do you want to use with this application?

DD Only: Use only roles and policies that are defined in the deployment descriptors.

Custom Roles: Use roles that are defined in the Administration Console; use policies that are defined in the deployment descriptor.

Custom Roles and Policies: Use only roles and policies that are defined in the Administration Console.

Advanced: Use a custom model that you have configured on the realm's configuration page.

Source accessibility

How should the source files be made accessible?

Use the defaults defined by the deployment's targets

Recommended selection.

Copy this application onto every target for me

During deployment, the files will be copied automatically to the managed servers to which the application is targeted.

I will make the deployment accessible from the following location

6. Select the **Deploy as Library** option and click **Finish**.
7. Confirm that the oracle.soa.workflow(11.1.1,11.1.1) library is in the **Active** state, as shown in [Figure 26–31](#).

Figure 26–31 Oracle WebLogic Server Administration Console: The oracle.soa.workflow Active State

<input type="checkbox"/>	oracle.soa.workflow(11.1.1,11.1.1)	Active	Library
<input type="checkbox"/>	oracle.soa.worklist(11.1.1,11.1.1)	Active	Library
<input type="checkbox"/>	oracle.wsm.seedpolicies(11.1.1,11.1.1)	Active	Library
<input type="checkbox"/>	OracleAppsAdapter	Installed	Resource Adapter
<input type="checkbox"/>	OracleBamAdapter	Installed	Resource Adapter

See [Section 26.8.5.2, "Defining the Foreign JNDI Provider on a non-SOA Oracle WebLogic Server,"](#) to continue.

26.8.5.2 Defining the Foreign JNDI Provider on a non-SOA Oracle WebLogic Server

Use Oracle WebLogic Server Administration Console to complete this portion of the task.

To define the foreign JNDI provider:

1. In the **Domain Structure** area, expand **Services** and click **Foreign JNDI Providers**.
2. Click **New**.
3. In the **Name** field, enter `ForeignJNDIProvider-SOA`, as shown in [Figure 26–32](#), and click **OK**.

Figure 26–32 *Creating a Foreign JNDI Provider*

Home > Summary of Deployments > Summary of Foreign JNDI Providers

Create a Foreign JNDI Provider

OK Cancel

Foreign JNDI Provider Properties

The following properties will be used to identify your new Foreign JNDI Provider.
* Indicates required fields

What would you like to name your new Foreign JNDI Provider?

* Name: ForeignJNDIProvider-SOA

OK Cancel

4. Click the **ForeignJNDIProvider-SOA** link.
5. Do the following and click **Save**.
 - For **Initial Context Factory**, enter `weblogic.jndi.WLInitialContextFactory`.
 - For **Provider URL**, enter `t3://soa_hostname:soa_portnumber/soa-infra`.
 - For **User**, enter `weblogic`.
 - For **Password**, enter `weblogic`.

[Figure 26–33](#) shows the page where you enter this information.

Figure 26–33 Defining the Foreign JNDI Provider

Save

A foreign JNDI provider represents a JNDI tree that can reside outside of a WebLogic Server. This could be a JNDI tree in a different server environment or within an external Java program. By setting up a foreign JNDI provider you can lookup and use an object that exists outside of the WebLogic server environment with the same ease that you would use an object bound in your WebLogic server instance. Use this page to configure a foreign JNDI provider.

Name:	ForeignJNDIProvider-SOA	The user-specified name of this MBean instance. More Info...
Initial Context Factory:	<input type="text"/>	The initial context factory to use to connect. This class name depends on the JNDI provider and the vendor that are being used. The value corresponds to the standard JNDI property, <code>java.naming.factory.initial</code> . More Info...
Provider URL:	<input type="text"/>	The foreign jndi provider url. This value corresponds to the standard JNDI property, <code>java.naming.provider.url</code> . More Info...
User:	<input type="text"/>	The remote server's user name. More Info...
Password:	<input type="text"/>	The remote server's user password. More Info...
Confirm Password:	<input type="text"/>	
Properties:	<input type="text"/>	Any additional properties that must be set for the JNDI provider. These properties will be passed directly to the constructor for the JNDI provider's <code>InitialContext</code> class. More Info...

See [Section 26.8.5.3, "Defining the Foreign JNDI Provider Links on a non-SOA Oracle WebLogic Server,"](#) to continue.

26.8.5.3 Defining the Foreign JNDI Provider Links on a non-SOA Oracle WebLogic Server

Use Oracle WebLogic Server Administration Console to complete this portion of the task.

To define the foreign JNDI provider links:

1. In the **Domain Structure** area, expand **Services** and click **Foreign JNDI Providers**.
2. Click the **ForeignJNDIProvider-SOA** link.
3. Click the **Links** tab.
4. Click **New**.

[Figure 26–34](#) shows the **Links** tab.

Figure 26–34 Defining the Foreign JNDI Provider Links: The Links Tab

Home > Summary of Deployments > Summary of Foreign JNDI Providers > ForeignJNDIProvider-SOA > Summary of Foreign JNDI Providers > ForeignJNDIProvider-SOA

Settings for ForeignJNDIProvider-SOA

Configuration Notes

General Links

This page lists existing ForeignJNDILinks associated with this ForeignJNDIProvider.

[Customize this table](#)

Foreign JNDI Links

New Clone Delete Showing 0 to 0 of 0 Previous | Next

<input type="checkbox"/>	Name	Local JNDI Name	Remote JNDI Name
There are no items to display			

New Clone Delete Showing 0 to 0 of 0 Previous | Next

5. Do the following and click OK.
 - For **Name**, enter RuntimeConfigService.
 - For **Local JNDI Name**, enter RuntimeConfigService.
 - For **Remote JNDI Name**, enter RuntimeConfigService.

[Figure 26–35](#) shows where you do this.

Figure 26–35 Defining the Foreign JNDI Provider Links: Link Properties

Home > Summary of Deployments > Summary of Foreign JNDI Providers > ForeignJNDIProvider-SOA > Summary of Foreign JNDI Providers > ForeignJNDIProvider-SOA

Create a Foreign JNDI Link

OK Cancel

Foreign JNDI Link Properties

The following properties will be used to identify your new Foreign JNDI Link.

* Indicates required fields

What would you like to name your new Foreign JNDI Link?

*Name:

Specify additional properties for this link

Local JNDI Name:

Remote JNDI Name:

OK Cancel

6. Do the following and click OK.

- For **Name, Local JNDI Name, Remote JNDI Name**, enter `ejb/bpel/services/workflow/TaskServiceBean`.
- For **Name, Local JNDI Name, Remote JNDI Name**, enter `ejb/bpel/services/workflow/TaskMetadataServiceBean`.
- For **Name, Local JNDI Name, Remote JNDI Name**, enter `TaskReportServiceBean`.
- For **Name, Local JNDI Name, Remote JNDI Name**, enter `TaskEvidenceServiceBean`.
- For **Name, Local JNDI Name, Remote JNDI Name**, enter `TaskQueryService`.
- For **Name, Local JNDI Name, Remote JNDI Name**, enter `UserMetadataService`.

See [Section 26.8.5.4, "Including a Grant for bpm-services.jar,"](#) to continue.

26.8.5.4 Including a Grant for bpm-services.jar

To include a grant for bpm-services.jar, edit the `system-jazn-data.xml` file and then restart the non-SOA Oracle WebLogic Server.

To include a grant for bpm-services.jar:

1. Locate the `system-jazn-data.xml` file by navigating to the domain directory, `soa-infra`, and then to

```
ORACLE_WEBLOGIC_INSTALL/user_projects/domains/your_domain_name/config/fmwconfig
```

2. In `system-jazn-data.xml`, add the following grant. (If all or some portion of the grant exists, then add only what is missing.)

```
<grant>
  <grantee>
    <codesource>
      <url>file: ORACLE_JDEV_HOME/jdeveloper/soa/modules/oracle.soa.workflow_
11.1.1/bpm-services.jar</url>
    </codesource>
  </grantee>
  <permissions>
    <permission>
      <class>oracle.security.jps.JpsPermission</class>
      <name>VerificationService.createInternalWorkflowContext</name>
    </permission>
    <permission>
      <class>oracle.security.jps.service.credstore.CredentialAccessPermission
</class>
      <name>credstoressp.credstore.BPM-CRYPTO.BPM-CRYPTO</name>
      <actions>read,write</actions>
    </permission>
    <permission>
      <class>oracle.security.jps.JpsPermission</class>
      <name>IdentityAssertion</name>
      <actions>*</actions>
    </permission>
  </permissions>
</grant>
```

3. Restart the non-SOA Oracle WebLogic Server.

See [Section 26.8.5.5, "Deploying the Application,"](#) to continue.

26.8.5.5 Deploying the Application

Deploy the application that contains the task display form to a non-SOA Oracle WebLogic Server the same way other applications are deployed. When you set up the application server connection, specify the domain on the non-SOA server (the domain you specified in Step 1 of [Section 26.8.5.4, "Including a Grant for bpm-services.jar."](#) See [Chapter 43, "Deploying SOA Composite Applications"](#) for information on deploying applications.

26.8.6 What Happens When You Deploy the Task Display Form

When the task form is deployed, an automatic association is created between the task metadata and the task flow application URL. Use Oracle Enterprise Manager 11g Fusion Middleware Control to update this mapping. Access the task flow component in the **Component Metrics** table for a specific SOA composite application. The Administration tab shows the URI for the task form. See *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite* for more information. If the task flow is configured for HTTPS access, you may need to do additional settings in Enterprise Manager.

See [Chapter 27, "Using Oracle BPM Worklist"](#) for information on how to act on tasks.

Note: If you want to access the task display form from a different URL that has a different port number than the hostname and port number previously set in Oracle WebLogic Server Administration Console, then you must change the port number for the front-end in Oracle WebLogic Server Administration Console and redeploy the task display form so that the task details appear correctly in the worklist.

26.9 Displaying a Task Display Form in the Worklist

The task display form is displayed in Oracle BPM Worklist, a web-based interface for users to act on their assigned human tasks. Specific actions are available or unavailable depending on a user's privileges.

[Figure 26-36](#) shows how the task display form for the help desk request example is displayed in the Worklist Application task details page.

Figure 26–36 Worklist Task Details Page

Help desk request for wfaulk

 Details

Contents

Location: California
 Type: Hardware
 Problem Description: Unable to reboot the system
 Severity:
 Status:

Requester

ID: wfaulk
 First Name: William
 Last Name: Faulkner
 Email: user1@us.oracle.com
 Phone: [0]

Resolution

Comment: None
 Resolved By: None

History

Task Snapshot Future Participants Full task actions

#	Participant	Action
1	 Stage 1	
1.1	 jstein	Assigned
1.2	 jcooper	RESOLVED, Outcome U
1.3	 jstein	RESOLVED, Outcome U
1.4	 demoadmin	RESOLVED, Alerted
1.5	 California	RESOLVED, Outcome U
1.6	 jcooper	RESOLVED, Outcome U
1.7	 jstein	RESOLVED, Outcome U
2	 jstein	RESOLVED, Completed

26.9.1 How To Display the Task Display Form in the Worklist

The task display form is available in Oracle BPM Worklist after you log in. See [Section 27.2.1, "How To Log In to the Worklist"](#) for instructions.

26.10 Displaying a Task in an Email Notification

Figure 26–37 shows how an email task notification appears in email.

Figure 26–37 Email Task Notification

Subject: Action Required:Help desk request for wfaulk
From: tom@maprao-pc.com
To: dhiraj@maprao-pc.com

Task Help desk request for wfaulk requires your attention.
 Please access the task in the [Worklist Application](#) or take direct action using the links below:

Actions: [RESOLVED](#) [UNRESOLVED](#)

  **Help desk request for wfaulk**

Task Number 200011
 State Assigned
 Outcome Assigned
 Priority 3
 Created Mar 16, 2009 6:14 PM
 Updated Mar 16, 2009 6:14 PM
 Expiration Date Mar 18, 2009 6:24 PM
 Assignees jstein

 **Contents**

Location California
 Type Hardware
 Problem Description Unable to reboot the system
 Severity 2
 Status Created

Requester
 ID wfaulk
 First Name William
 Last Name Faulkner
 Email user1@us.oracle.com
 Phone 2222222222

Resolution
 Comment None
 Resolved By None

 **Comments**

No rows available

You can click an available action, **RESOLVED** or **UNRESOLVED**, or click the **Worklist Application** link to log in to the worklist. Clicking an action displays an email composer window in which you can add a comment and send the email.

Using Oracle BPM Worklist

This chapter describes how worklist users and administrators interact with Oracle BPM Worklist, and how to customize the worklist display to reflect local business needs, languages, and time zones.

This chapter contains the following topics:

- [Section 27.1, "Introduction to Oracle BPM Worklist"](#)
- [Section 27.2, "Logging In to Oracle BPM Worklist"](#)
- [Section 27.3, "Customizing the Task List Page"](#)
- [Section 27.4, "Acting on Tasks: The Task Details Page"](#)
- [Section 27.5, "Approving Tasks"](#)
- [Section 27.6, "Setting a Vacation Period"](#)
- [Section 27.7, "Setting Rules"](#)
- [Section 27.8, "Using the Worklist Administration Functions"](#)
- [Section 27.9, "Specifying Notification Settings"](#)
- [Section 27.10, "Using Flex Fields"](#)
- [Section 27.11, "Creating Worklist Reports"](#)
- [Section 27.12, "Accessing Oracle BPM Worklist in Local Languages"](#)

See [Chapter 28, "Building a Custom Worklist Client"](#) for how to use the APIs exposed by the workflow service.

27.1 Introduction to Oracle BPM Worklist

Oracle BPM Worklist enables business users to access and act on tasks assigned to them. For example, from a worklist, a loan agent can review loan applications or a manager can approve employee vacation requests. These processes are defined in human tasks.

Oracle BPM Worklist provides different functionality based on the user profile. Standard user profiles include task assignee, supervisor, process owner, and administrator. For example, worklist users can update payloads, attach documents or comments, and route tasks to other users, in addition to completing tasks by providing conclusions such as approvals or rejections. Supervisors or group administrators can use the worklist to analyze tasks assigned to a group and route them appropriately.

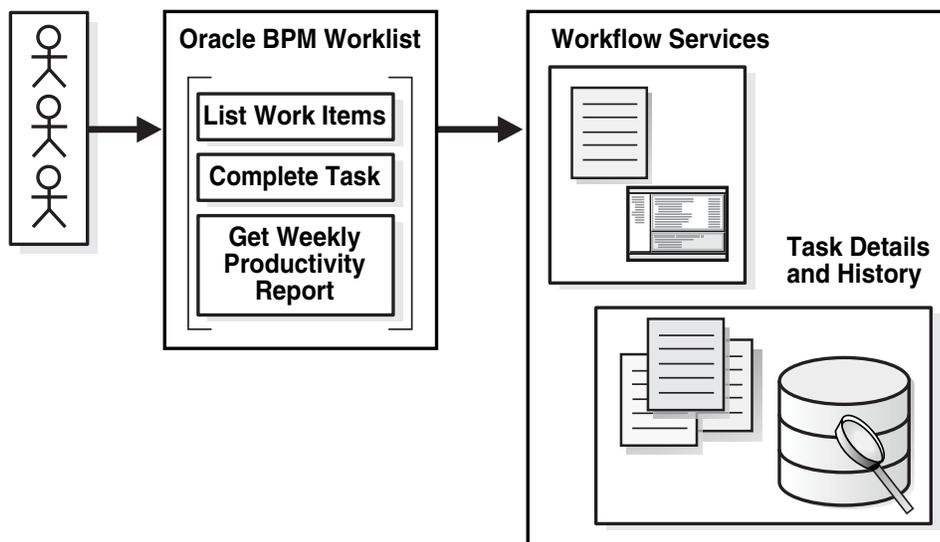
Users can customize their task lists, as required, by adding worklist views, for example, selecting the columns to display, or displaying a subset of the tasks based on filter criteria.

Using Oracle BPM Worklist, task assignees can do the following:

- Perform authorized actions on tasks in the worklist, acquire and check out shared tasks, define personal to-do tasks, and define subtasks.
- Filter tasks in a worklist view based on various criteria.
- Work with standard work queues, such as high priority tasks, tasks due soon, and so on. Work queues allow users to create a custom view to group a subset of tasks in the worklist, for example, high priority tasks, tasks due in 24 hours, expense approval tasks, and more.
- Define custom work queues.
- Gain proxy access to part of another user's worklist.
- Define custom vacation rules and delegation rules.
- Enable group owners to define task dispatching rules for shared tasks.
- Collect a complete workflow history and audit trail.
- Use digital signatures for tasks.

Figure 27–1 shows an illustration of Oracle BPM Worklist.

Figure 27–1 Oracle BPM Worklist—Access Tasks, Forms, Attachments, and Reports



The worklist is rendered in a browser by a task display form that you create using ADF task flows in Oracle JDeveloper. See [Chapter 26, "Designing Task Display Forms for Human Tasks"](#) for more information.

Users can also act on tasks through portals such as Oracle WebCenter. Portals enable users to present information from multiple, unrelated data sources in a single organized view. This view, a portal page, can contain one or more components called portlets that can each collect content from different data sources.

You can build clients for workflow services using the APIs exposed by the workflow service. The APIs enable clients to communicate with the workflow service using local and remote EJBs, SOAP, and HTTP.

27.1.1 What You May Need To Know About Oracle BPM Worklist

Note the following:

- Only one identity provider is supported. Java policy store does not support multiple providers in a sequence. Therefore, fall-through from one directory server to another is not supported for worklists.

27.2 Logging In to Oracle BPM Worklist

Table 27-1 lists the different types of users recognized by Oracle BPM Worklist, based on the privileges assigned to the user.

Table 27-1 Worklist User Types

Type of User	Access
End user (user)	Acts on tasks assigned to him or his group and has access to system and custom actions, routing rules, and custom views
Supervisor (manager)	Acts on the tasks, reports, and custom views of his reportees, in addition to his own end-user access
Process owner	Acts on tasks belonging to the process but assigned to other users, in addition to his own end-user access
Group administrator	Manages group rules and dynamic assignments, in addition to his own end-user access
Workflow administrator	Administers tasks that are in an errored state, for example, tasks that must be reassigned or suspended. The workflow administrator can also change application preferences and map flex fields, and manage rules for any user or group, in addition to his own end-user access.

Note: Multiple authentication providers (for example, SSO and forms) are not supported.

27.2.1 How To Log In to the Worklist

To log in, you must have installed Oracle SOA Suite and the SOA server must be running. See *Oracle Fusion Middleware Installation Guide for Oracle SOA Suite* for more information.

Use a supported web browser:

- Microsoft Internet Explorer 7.x
- Mozilla Firefox 2.x
- Mozilla Firefox 3.x

To log in:

1. Go to

`http://host_name:port_number/integration/worklistapp`

- `host_name` is the name of the host computer on which Oracle SOA Suite is installed
 - The `port_number` used at installation
2. Enter the user name and password.

You can use the preseeded user to log in as an administrator. If you have loaded the demo user community in the identity store, then you can use other users such as `jstein` or `jcooper`.

The user name and password must exist in the user community provided to JAZN. See *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite* for the organizational hierarchy of the demo user community used in examples throughout this chapter.

3. Click **Login**.

27.2.1.1 Enabling the `weblogic` User for Logging in to the Worklist

For the `weblogic` user in OID to log in to Oracle BPM Worklist, the OID Authenticator must have an Administrators group, and the `weblogic` user must be a member of that group.

To enable the `weblogic` user:

1. Create a `weblogic` user in OID using the LDAP browser. The `users.ldif` file is imported to OID as follows:

```
dn: cn=weblogic,cn=Users,dc=us,dc=oracle,dc=com
objectclass: inetorgperson
objectclass: organizationalPerson
objectclass: person
objectclass: orcluser
objectclass: orcluserV2
objectclass: top
sn: weblogic
userpassword: welcome1
uid: weblogic
```

2. Create an Administrators group in OID and assign the `weblogic` user to it. The `groups.ldif` file is imported to OID as follows:

```
dn: cn=Administrators,cn=Groups,dc=us,dc=oracle,dc=com
objectclass: groupOfUniqueNames
objectclass: orclGroup
objectclass: top
owner: cn=orcladmin,cn=Users,dc=us,dc=oracle,dc=com
uniquemember: cn=weblogic,cn=Users,dc=us,dc=oracle,dc=com
```

27.2.2 What Happens When You Log In to the Worklist

Identity service workflow APIs authenticate and authorize logins using a user name, password, and optionally a realm set, if multiple realms were defined for an organization. See [Section 27.8.2, "How To Set the Worklist Display \(Application Preferences\)"](#) for information on how administrators can set a preference to change the realm label displayed in the interface, or specify an alternative location for the source of the login page image.

After a user logs in, the **Home** (task list) page displays tasks for the user based on the user's permissions and assigned groups and roles. The **My Tasks** tab and the **Inbox** are displayed by default. The actions allowed from the **Actions** list also depend on the logged-in user's privileges.

[Figure 27–2](#) shows an example of the **Home** page.

Figure 27–2 Oracle BPM Worklist—The Home (Task List) Page

The screenshot displays the Oracle BPM Worklist interface. At the top, the header includes the Oracle logo and 'BPM Worklist', along with navigation links for 'Home', 'Reports', and 'Preferences'. Below this, a set of tabs allows users to switch between 'My Tasks', 'Initiated Tasks', 'My Staff Tasks', and 'Administration Tasks'. The left sidebar contains 'Worklist Views' with options like 'Inbox', 'My Work Queues', 'Standard Views', 'My Views', and 'Proxy Work Queues'. Below this is a 'Task Status' section showing counts for various task states: Assigned (4), Completed (5), Suspended, Withdrawn, Expired, Errored, Alerted, Info, Requested, Deleted, Stale, and Total (9). The main area features a table of tasks with columns for Title, Number, Priority, Assignees, State, Created, and Expires. A task titled 'Help desk request for wfaulk' is selected, showing its details: Assignees (jstein), Expiration date (Mar 25, 2009 11:19 AM), State (Assigned), Creator (wfaulk), Acquired By (wfaulk), Created (Mar 24, 2009 11:09 AM), Task Number (200030), Updated (Mar 24, 2009 11:09 AM), and Priority (3). The 'Contents' section provides details on Location (California), Type (Hardware), Problem Description (Unable to reboot the system), Severity (2), and Status (Created). The 'Requester' section lists ID (wfaulk), First Name (William), Last Name (Faulkner), Email (user1@us.oracle.com), and Phone ([]). The 'Resolution' section shows Comment (None) and Resolved By (None). The 'History' section includes checkboxes for 'Task Snapshot', 'Future Participants', and 'Full task actions', followed by a table of task actions with columns for #, Participant, Action, and Action Date.

Title	Number	Priority	Assignees	State	Created	Expires
todo	200005	1	jstein (U)	Assigned	Mar 5, 2009 6:38 PM	
Help desk request for wfaulk	200028	3	jstein (U)	Assigned	Mar 23, 2009 3:24 PM	Mar 24,
Help desk request for wfaulk	200029	3	jstein (U)	Assigned	Mar 23, 2009 4:22 PM	Mar 24,
Help desk request for wfaulk	200030	3	jstein (U)	Assigned	Mar 24, 2009 11:09 AM	Mar 25,

#	Participant	Action	Action Date
1	Stage 1		
1.1	jstein	Assigned	Mar 24, 2009 11:09 AM

Table 27–2 describes the components of the Home (task list) page.

Table 27–2 Components of the Home (Task List) Page

Component	Description
<p>Tabs</p>	<p>The tabs displayed depend on the role granted to the logged-in user.</p> <ul style="list-style-type: none"> ▪ Everyone (the user role) sees My Tasks and Initiated Tasks. ▪ Users who are also managers see the My Tasks, Initiated Tasks, and My Staff Tasks tabs. ▪ Users who are also owners (of a process) see the My Tasks, Initiated Tasks, and Administration Tasks tabs. ▪ Users who are also administrators (the BPMWorkflowAdmin), but not managers, see the My Tasks, Initiated Tasks, Administration Tasks, Administration, Evidence Search, and Approval Groups tabs. ▪ Users who are managers and administrators see all the tabs— My Tasks, Initiated Tasks, My Staff Tasks, Administration Tasks, Administration, Evidence Search, and Approval Groups. ▪ Users with the workflow.admin.evidenceStore permission also see the Evidence Search tab. <p>See the following for more information:</p> <ul style="list-style-type: none"> ▪ Section 27.4.4, "How To Act on Tasks That Require a Digital Signature," for information about evidence search ▪ Section 27.8.1, "How To Manage Other Users' or Groups' Rules (as an Administrator)"
<p>Worklist Views</p>	<p>Inbox, My Work Queues, Proxy Work Queues—See Section 27.3.2, "How To Create and Customize Worklist Views," for more information.</p>
<p>Task Status</p>	<p>A bar chart shows the status of tasks in the current view. See Section 27.3.3, "How To Customize the Task Status Chart," for more information.</p>
<p>Display Filters</p>	<p>Specify search criteria from the View, Assignee or Status fields. The category filters that are available depend on which tab is selected.</p> <ul style="list-style-type: none"> ▪ The View filters are Inbox, Due Soon, High Priority, and New Tasks. ▪ From the My Tasks tab, the Assignee filters are My, Group, My & Group, Previous (tasks worked on in the past), and Reviewer. From the Initiated Tasks tab, the only assignee filter is Creator. From the My Staff Tasks tab, the only assignee filter is Reportees. From the Administration Tasks tab, the only assignee filter is Admin. ▪ The Status filters include Any, Assigned, Completed, Suspended, Withdrawn, Expired, Errored, Alerted, Information Requested. <p>Use Search to enter a keyword, or use Advanced Search. See Section 27.3.1, "How To Filter Tasks," for more information.</p>
<p>Actions List</p>	<p>Select a group action (Claim) or a custom action (for example, Approve or Reject) that was defined for the human task. Claim appears for tasks assigned to a group or multiple users; one user must claim the task before it can be worked. Other possible actions for a task, such as system actions, are displayed on the task details page for a specific task. You can also create ToDo tasks and subtasks here.</p>
<p>Default Columns</p>	<p>Title—The title specified when the human task was created. Tasks associated with a purged or archived process instance do not appear.</p> <p>Number—The task number generated when the BPEL process was created.</p> <p>Priority—The priority specified when the human task was created. The highest priority is 1; the lowest is 5.</p> <p>Assignees—The user or group or application roles.</p> <p>State—Select from Assigned, Completed, Errored, Expired, Information Requested, Stale, Suspended, or Withdrawn.</p> <p>Created—Date and time the human task was created</p> <p>Expires—Date and time the tasks expires, specified when the human task was created</p>
<p>Task Details</p>	<p>The lower section of the worklist displays the inline view of the task details page. Buttons indicate available actions. See Section 27.4, "Acting on Tasks: The Task Details Page," for more information.</p>

Figure 27-2 also shows the **Administration**, **Reports**, and **Preferences** links (upper-right corner). Table 27-3 summarizes the **Home**, **Administration**, **Reports**, and **Preferences** pages.

Table 27-3 Worklist Main Pages Summary

Page	Description
Home	As described in Table 27-2, the logged-in user's list of tasks, details for a selected task, and all the functions needed to start acting on a task are provided.
Administration	The following administrative functions are available: <ul style="list-style-type: none"> ■ Setting application preferences ■ Mapping flex fields ■ Searching the evidence store ■ Specifying approval group ■ Configuring tasks
Reports	The following reports are available: Unattended Tasks Report, Tasks Priority Report, Tasks Cycle Time Report, Tasks Productivity Report, and Tasks Time Distribution Report. See Section 27.11.1, "How To Create Reports," for more information.
Preferences	Preference settings include: <ul style="list-style-type: none"> ■ Setting rules for users or groups, including vacation rules, and setting vacation periods ■ Uploading certificates ■ Specifying user notification channels and message filters

27.3 Customizing the Task List Page

You can customize your task list in several ways, including adding worklist views, selecting which columns to display, and displaying a subset of the tasks based on filter criteria. Resize the task list display area to increase the number of tasks fetched.

27.3.1 How To Filter Tasks

Figure 27-3 shows the filter fields.

Figure 27-3 Filters—Assignee, Status, Search, and Advanced Search

Title	Number	Priority	Assignees	State	Created
Vacation Request for jcooper	200203	3	jstein (U)	Assigned	Mar 16, 2009 2:16 PM

Filters are used to display a subset of tasks, based on the following filter criteria:

- **Assignee**—Select from the following:
 - **My**—Retrieves tasks directly assigned to the logged-in user
 - **Group**—Retrieves the following:
 - * Tasks that are assigned to groups that the logged-in user belongs to
 - * Tasks that are assigned to an application role that the logged-in user is assigned

* Tasks that are assigned to multiple users, one of which is the logged-in user

- **My & Group**—Retrieves all tasks assigned to the user, whether through direct assignment, or by way of a group, application role, or list of users
- **Previous**—Retrieves tasks that the logged-in user has updated
- **Reviewer**—Retrieves task for which the logged-in user is a reviewer
- **Status**—Select from the following: **Any**, **Assigned**, **Completed**, **Suspended** (can be resumed later), **Withdrawn**, **Expired**, **Errored** (while processing), **Alerted**, or **Information Requested**.
- **Search**—Enter a keyword to search task titles, comments, identification keys, and the flex string fields of tasks that qualify for the specified filter criterion.
- **Advanced**—Provides additional search filters.

To filter tasks based on priority, assignee, or status:

1. Select any combination of options from the **Priority**, **Assignee**, or **Status** lists.
2. Click **Refresh**.

To filter tasks based on keyword search:

1. Enter a keyword to search task titles, comments, identification keys, and the flex string fields of tasks that qualify for the specified filter criterion.
2. Click **Refresh**.

To filter tasks based on an advanced search:

Flex field attribute labels can be used in an advanced search if you select task types for which flex field mappings have been defined.

1. Click **Advanced**.
2. (Optional) Check **Save As View**, provide a view name, and use the **Display** tab to provide other information, as shown in [Figure 27-4](#) and [Figure 27-5](#).

Figure 27-4 Worklist Advanced Search—Definition Tab

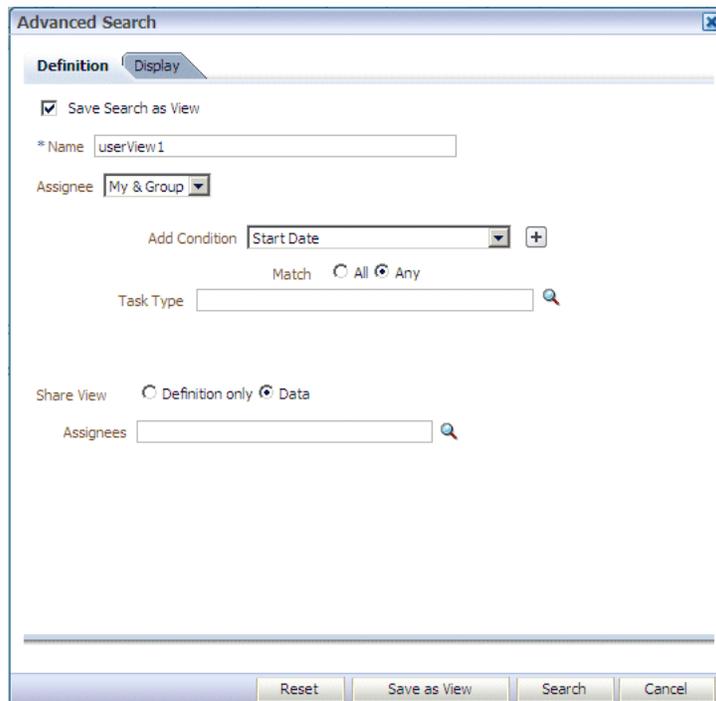


Figure 27-5 Worklist Advanced Search—Display Tab

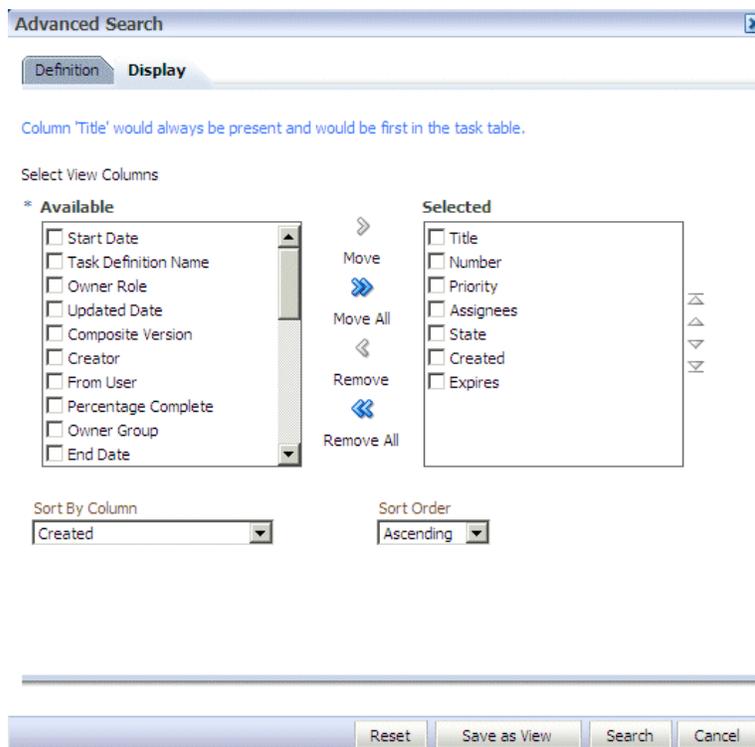


Table 27-4 describes the advanced search view columns available in the **Display** tab.

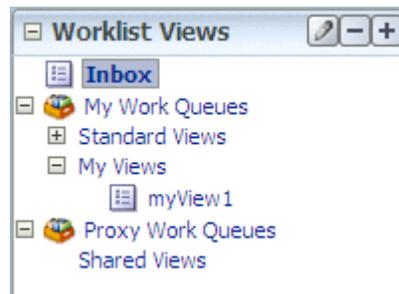
Table 27-4 Advanced Search—View Columns

Column	Description
Start Date	The start date of the task (used with ToDo tasks).
Task Definition Name	The name of the task component that defines the task instance.
Owner Role	The application role (if any) that owns the task instance. Task owners can be application roles, users, or groups. If the owner of the task is an application role, this field is set.
Updated Date	The date the task instance was last updated.
Composite Version	The version of the composite that contains the task component that defines the task instance.
Creator	The name of the creator of the task.
From User	The from user for the task.
Percentage Complete	The percentage of the task completed (used with ToDo tasks).
Owner Group	The group (if any) that owns the task instance. Task owners can be application roles, users, or groups. If the owner of the task is a group, this field is set.
End Date	The end date of the task (used with ToDo tasks).
Composite	The name of the composite that contains the task component that defines the task instance.
Due Date	The due date of the task (used with ToDo tasks).
Composite Distinguished Name	The unique name for the particular deployment of the composite that contains the task component that defines the task instance.
Task Display URL	The URL to display the details for the task.
Updated By	The user who last updated the task.
Outcome	The outcome of the task, for example Approved or Rejected. This is only set on completed task instances.
Task Namespace	A namespace that uniquely defines all versions of the task component that defines this task instance. Different versions of the same task component can have the same namespace, but no two task components can have the same namespace.
Approvers	The approvers of the task.
Application Context	The application to which any application roles associated with the tasks (such as assignees, owners, and so on) belong.
Owner User	The user (if any) that owns the task instance. Task owners can be application roles, users, or groups. If the owner of the task is a user, this field is set.
Identifier	The (optional) custom unique identifier for the task. This is an additional unique identifier to the standard task number.
Category	The category of the task.
Acquired By	The name of the user who claimed the task in the case when the task is assigned to a group, application role, or to multiple users, and then claimed by the user.
Component	The name of the task component that defines the task instance.
Original Assignee User	The name of the user who delegated the task in the case when the user delegates a task to another user.
Assigned	The date that this task was assigned.

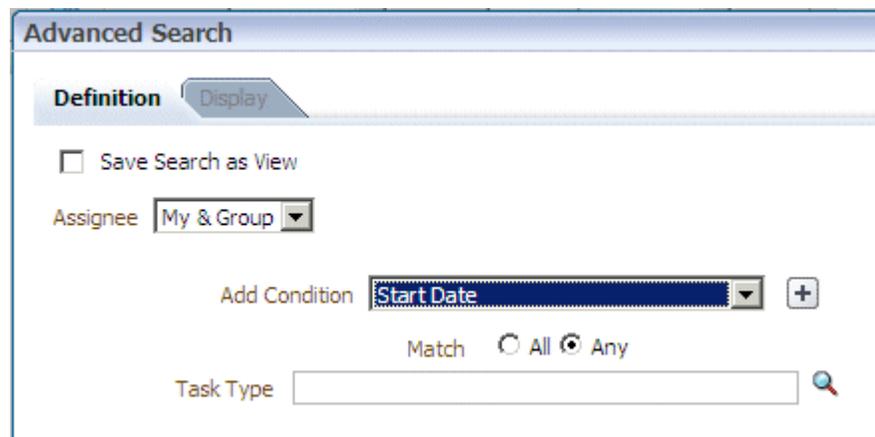
Table 27-4 (Cont.) Advanced Search—View Columns

Column	Description
Domain	The domain to which the composite that contains the task component that defines the task instance belongs.
Title	The title of the task.
Number	An integer that uniquely identifies the task instance.
Priority	An integer that defines the priority of the task. A lower number indicates a higher priority—typically numbers 1 to 5 are used.
Assignees	The current task assignees (users, groups or application roles).
State	The state of the task instance.
Created	The date that the task instance was created.
Expires	The date on which the task instance expires.

The saved view appears in the **Inbox** under **My Views**.



3. Select an assignee, as shown in [Figure 27-6](#).

Figure 27-6 Worklist Advanced Search

4. Add conditions (filters), as shown in [Figure 27-7](#).

Figure 27–7 Adding Filters for an Advanced Search on Tasks

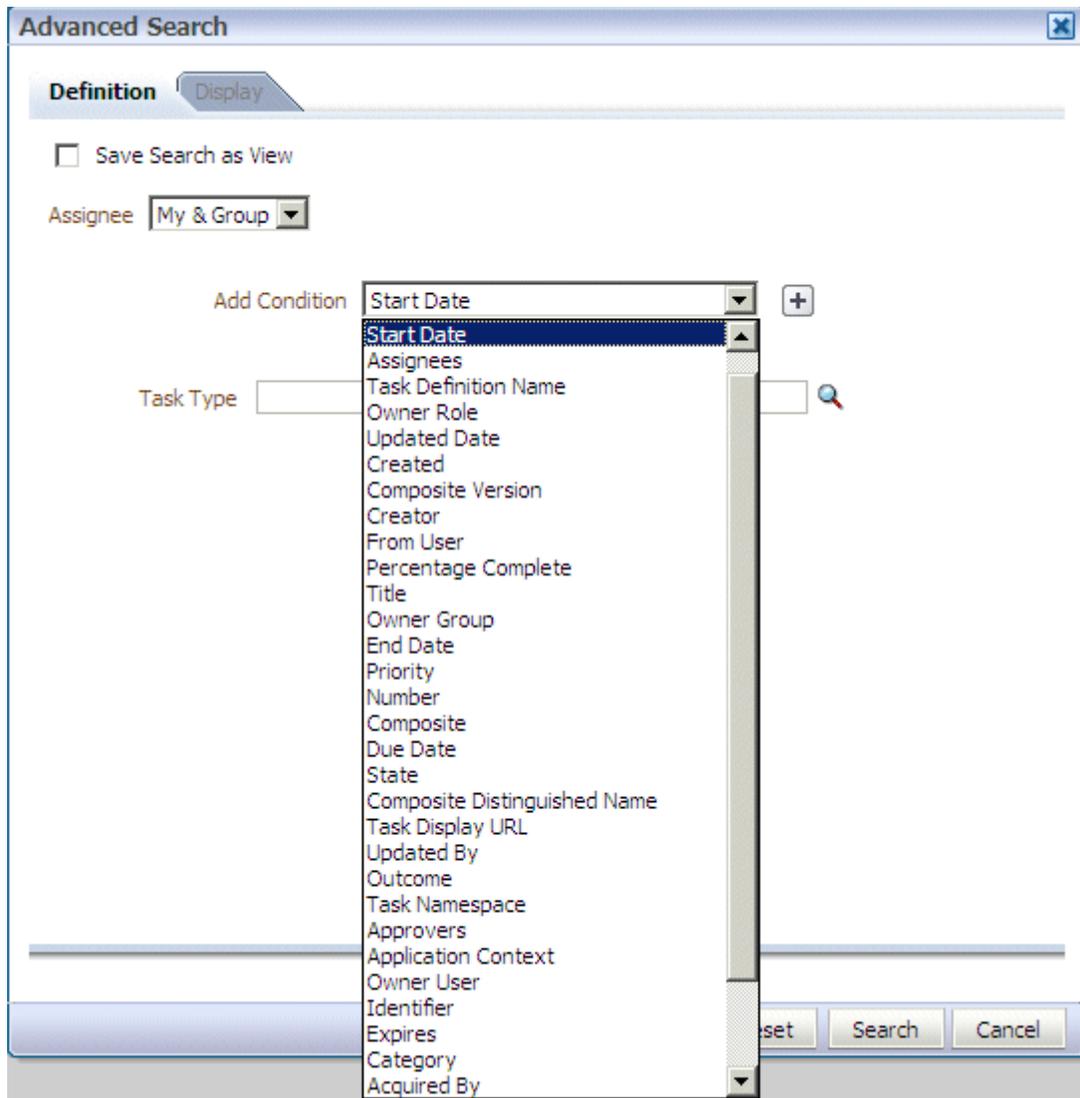


Table 27–5 describes the available conditions.

Table 27–5 Advanced Search—Conditions

Condition	Description
Start Date	The start date of the task (used with ToDo tasks).
Assignees	The current task assignees (users, groups or application roles).
Task Definition Name	The name of the task component that defines the task instance.
Owner Role	The application role (if any) that owns the task instance. Task owners can be application roles, users, or groups. If the owner of the task is an application role, this field is set.
Updated Date	The date that the task instance was last updated.
Created	The date that the task instance was created.
Composite Version	The version of the composite that contains the task component that defines the task instance.
Creator	The name of the creator of the task.

Table 27-5 (Cont.) Advanced Search—Conditions

Condition	Description
From User	The from user for the task.
Percentage Complete	The percentage of the task completed (used with ToDo tasks).
Title	The title of the task.
Owner Group	The group (if any) that owns the task instance. Task owners can be application roles, users, or groups. If the owner of the task is a group, this field is set.
End Date	The end date of the task (used with ToDo tasks).
Priority	An integer that defines the priority of the task. A lower number indicates a higher priority—typically numbers 1 to 5 are used.
Number	An integer that uniquely identifies the task instance.
Composite	The name of the composite that contains the task component that defines the task instance.
Due Date	The due date of the task (used with ToDo tasks).
State	The state of the task instance.
Composite Distinguished Name	The unique name for the particular deployment of the composite that contains the task component that defines the task instance.
Task Display URL	The URL to display the details for the task.
Updated By	The user who last updated the task.
Outcome	The outcome of the task, for example Approved or Rejected. This is only set on completed task instances.
Task Namespace	The namespace of the task.
Approvers	The approvers of the task.
Application Context	The application to which any application roles associated with the tasks (such as assignees, owners, and so on) belong.
Owner User	The user (if any) that owns the task instance. Task owners can be application roles, users, or groups. If the owner of the task is a user, this field is set.
Identifier	The (optional) custom unique identifier for the task. This is an additional unique identifier to the standard task number.
Expires	The date on which the task instance expires.
Category	The category of the task.
Acquired By	The name of the user who claimed the task in the case when the task is assigned to a group, application role, or to multiple users, and then claimed by the user.
Component	The name of the task component that defines the task instance.
Original Assignee User	The name of the user who delegated the task in the case when the user delegates a task to another user.
Assigned	The date that this task was assigned.
Domain	The domain to which the composite that contains the task component that defines the task instance belongs.

5. Add parameter values, shown in [Figure 27-8](#).

Figure 27–8 Advanced Search

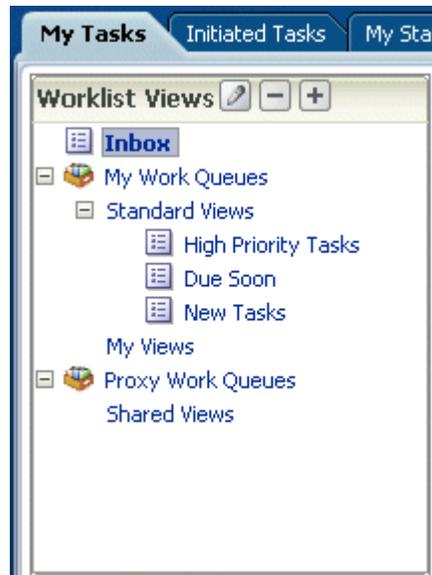
6. Select **Any** or **All** for matching multiple filters.
7. (Optional) Search on a task type.
8. Click **Search**.

The task list page with the tasks filtered according to your criteria appears.

27.3.2 How To Create and Customize Worklist Views

The **Worklist Views** area, shown in [Figure 27–9](#), displays the following:

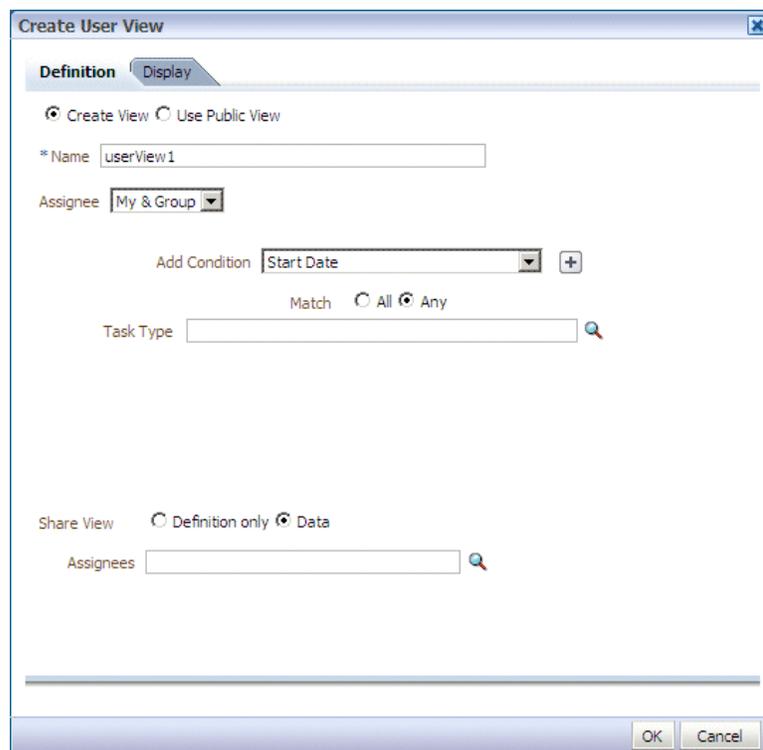
- **Inbox**—Shows all tasks that result from any filters you may have used. The default shows all tasks.
- **My Work Queues**—Shows standard views and views that you defined.
- **Proxy Work Queues**—Shows shared views.

Figure 27–9 Worklist Views

Use **Worklist Views** to create, share, and customize views.

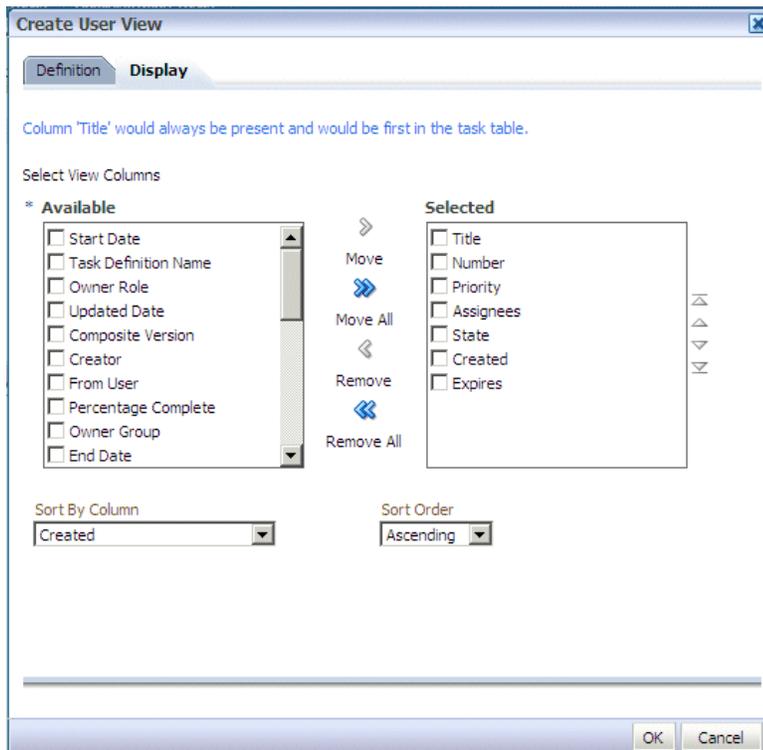
To create a worklist view:

1. In the **Worklist Views** section, click **Add**.
2. Use the **Definition** tab of the Create User View dialog, shown in [Figure 27–10](#).

Figure 27–10 Creating a Worklist View

- **Create View or Use Public View**—Create your own view or browse for a public view to copy.
 - **Name**—Specify a name for your view.
 - **Add to Standard Views**—This option applies to Administrators only. Administrators select this option to create the view as a standard view, which then appears in the **Standard Views** list for *all* worklist users.
 - **Assignee**—Select **My**, **Group**, **My&Group**, **Previous**, or **Reviewer**.
 - **Add Condition**—Select a filter from the list and click **Add**. For example, if you select **startDate**, and click **Add**, then a calendar and a list including **on**, **equals**, **not equals**, **greater than**, **less than**, and so on appears.
 - **Task Type**—Browse for a task type or leave the field blank for all types. Flex field attribute labels can be selected in the query and display columns dialogs if the selected task types have flex field mappings defined.
 - **Match**—Select **All** or **Any** to match the conditions you added.
 - **Share View**—You can grant access to another user to either the definition of this view, in which case the view conditions are applied to the grantee’s data, or to the data itself, in which case the grantee can see the grantor’s worklist view, including the data. Sharing a view with another user is similar to delegating all tasks that correspond to that view to the other user; that is, the other user can act on your behalf. Shared views are displayed under **Proxy Work Queues**.
 - **Assignees**—Specify the users (grantees) who can share your view.
3. Use the **Display** tab of the Create User View dialog, shown in [Figure 27–11](#), to customize the fields that appear in the view.

Figure 27–11 *Displaying Fields in a Worklist View*



- **Select View Columns**—Specify which columns you want to display in your task list. They can be standard task attributes or flex fields that have been mapped for the specific task type. The default columns are the same as the columns in your inbox.
 - **Sort by Column**—Select a column to sort on.
 - **Sort Order**—Select ascending or descending order.
4. Click **OK**.

To customize a worklist view:

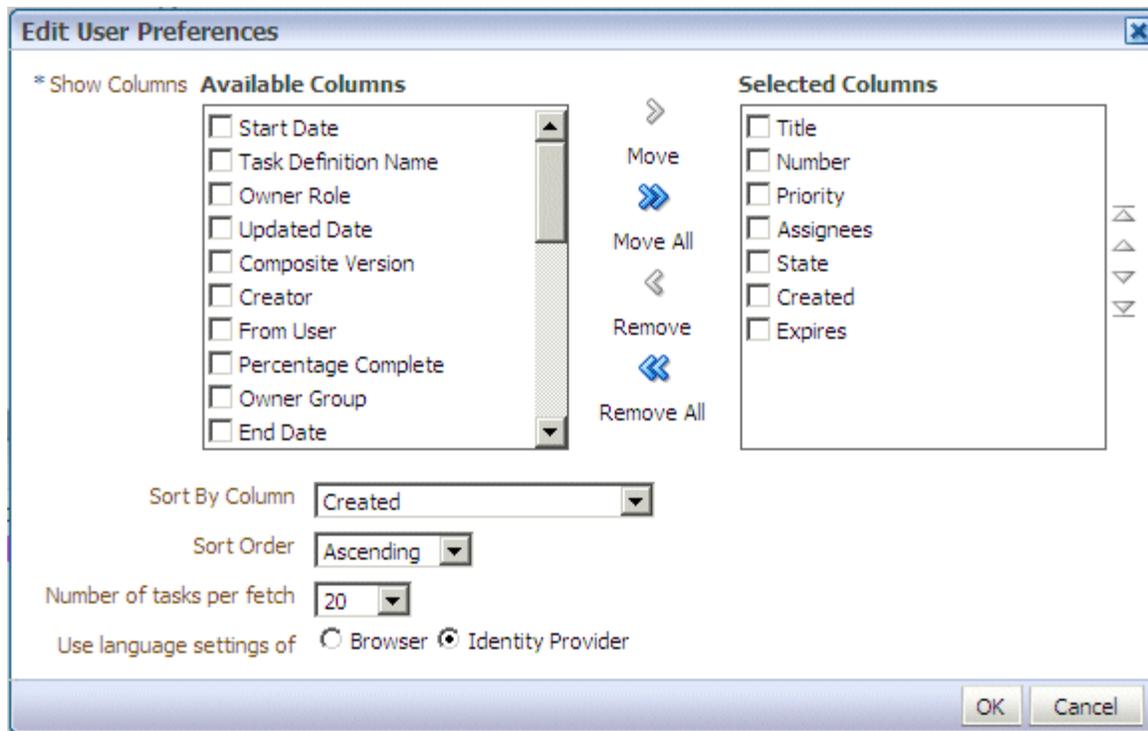
1. In the **Worklist Views** section, click the view name that you want to edit.
2. Click the **Edit** icon.
3. Use the **Definition** and **Display** tabs of the Edit User View dialog to customize the view, as shown in [Figure 27–12](#) and [Figure 27–13](#), and click **OK**.

Figure 27–12 Customizing a Worklist View

The screenshot shows the 'Edit User View : myView1' dialog box with the 'Definition' tab selected. The fields are as follows:

- *Name:** myView1
- Assignee:** My
- Add Condition:** Start Date
- Match:** All (selected), Any
- Task Type:** (empty)
- Assignees:** equals, j
- Share View:** Definition only, Data (selected)
- Assignees:** (empty)

Buttons for 'OK' and 'Cancel' are located at the bottom right of the dialog.

Figure 27–13 Customizing Fields in a Worklist View

When you select and move items from the **Available Columns** list to the **Selected Columns** list (or vice-versa), the items remain checked. Therefore, if you select items to move back, the previously selected items are also moved. Be sure to uncheck items after moving them between the lists if you intend to move additional columns.

27.3.3 How To Customize the Task Status Chart

The bar chart shows tasks broken down by status, with a count of how many tasks in each status category. The chart applies to the filtered set of tasks within the current view.

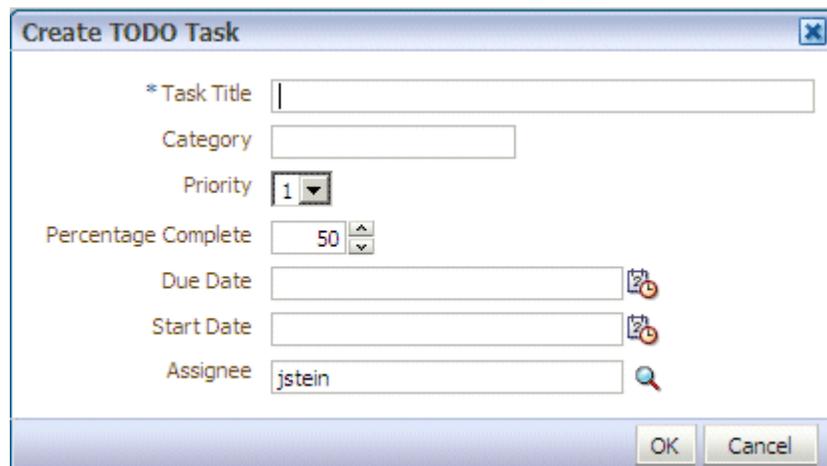
To customize the task status chart:

1. Click the **Edit** icon.
2. Add or remove status states for display, as shown in [Figure 27–14](#), and click **OK**.

Figure 27–14 Customizing the Task Status Chart

27.3.4 How To Create a ToDo Task

Use the Create ToDo Task dialog, shown in [Figure 27–15](#), to create a top-level ToDo task for yourself or others. This task is not associated with a business task.

Figure 27–15 The Create ToDo Task Dialog

ToDo tasks appear in the assignee's **Inbox**.

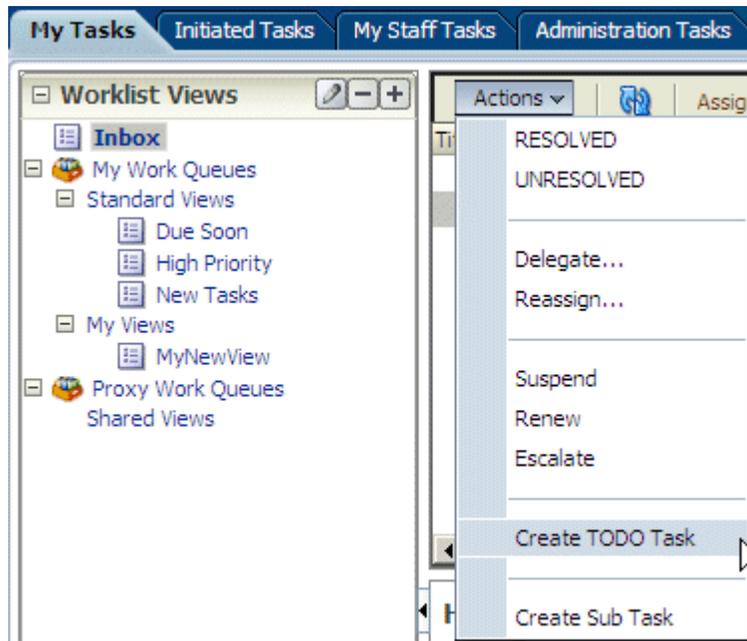
You can create ToDo tasks that are children of other ToDo tasks or business tasks. When all child ToDo tasks are 100% complete, the parent ToDo task is also marked as completed. If the parent ToDo task is completed, then child ToDo tasks are at 100% within the workflow system. If the parent is a business task, the child ToDo is not marked as completed. You must set the outcome and complete it. If you explicitly set a ToDo task to 100%, there is no aggregation on the parent task.

ToDo tasks can be reassigned, escalated, and so on, and deleted (logical delete) and purged (physical delete). Reassignment, escalation, and so on of the parent task does not affect the assignment of any child ToDo tasks. The completion percentage of a ToDo task can be reset to less than 100% after it is completed.

Assignment rules (such as vacation rules) are not applied to ToDo tasks. You cannot specify business rules for ToDo tasks.

To create a ToDo task:

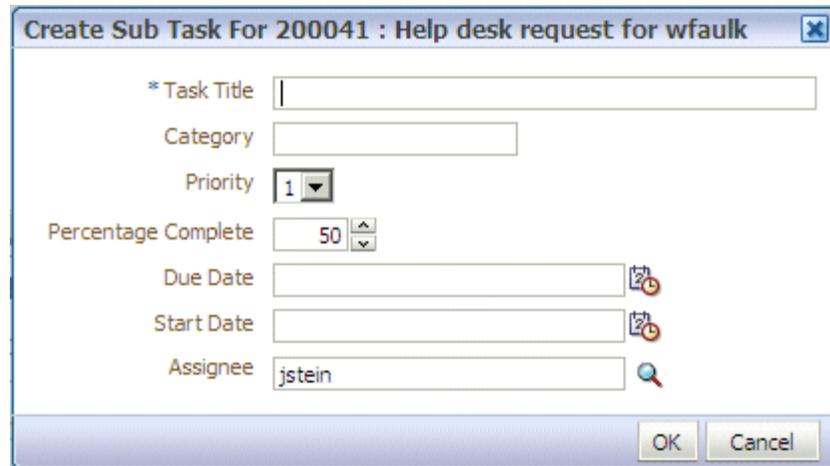
1. From the **Actions** list, select **Create TODO Task**, as shown in [Figure 27-16](#).

Figure 27-16 *Creating a ToDo Task*

2. Provide details in the Create ToDo Task dialog, shown in [Figure 27-16](#), and click **OK**.
 - **Task Title:** Enter anything that is meaningful to you.
 - **Category:** Enter anything that is meaningful to you.
 - **Priority:** Select from 1 (highest) to 5 (lowest)
 - **Percentage Complete:** This attribute indicates how much of the task is completed. 100% sets the attribute as completed.
 - **Due Date:** The due date does not trigger an expiration. You can also see overdue tasks. The start date need not be the current date.
 - **StartDate:** The task start date.
 - **Assignee:** You can assign yourself or someone else.

27.3.5 How To Create a Subtask

Use the Create Sub Task dialog, shown in [Figure 27-17](#), to create a subtask, which is a ToDo task for a business task. You must select a business task before selecting the **Create Sub Task** option (shown in [Figure 27-16](#)).

Figure 27–17 Creating a Subtask

The screenshot shows a dialog box titled "Create Sub Task For 200041 : Help desk request for wfaulk". It contains the following fields and controls:

- * Task Title: An empty text input field.
- Category: An empty text input field.
- Priority: A dropdown menu with "1" selected.
- Percentage Complete: A spinner control with "50" in the center.
- Due Date: An empty date input field with a calendar icon.
- Start Date: An empty date input field with a calendar icon.
- Assignee: A text input field containing "jstein" and a search icon.
- Buttons: "OK" and "Cancel" buttons at the bottom right.

Subtasks can be used to break down a business task into measurable subtasks, and can be created for ToDo tasks also. Multiple levels of subtasks are not supported (that is, you cannot have subtasks inside of subtasks). If you create multiple levels of subtasks, and attempt to act on the main task (for example, to approve or reject), you receive an error.

27.4 Acting on Tasks: The Task Details Page

Task details can be viewed inline (see the lower section in [Figure 27–2, "Oracle BPM Worklist—The Home \(Task List\) Page"](#)) or in a pop-up browser window. (Double-click the task.)

[Figure 27–18](#) shows the task details page.

Figure 27–18 Task Details Page

Help desk request for wfault

Task Actions ▼
 RESOLVED
UNRESOLVED

Details

Assignees

Creator

Created Apr 6, 2009 4:43 PM

Updated Apr 6, 2009 4:43 PM

Expiration date Apr 7, 2009 4:53 PM

Acquired By

Task Number 200040

Priority

State Assigned

Contents

Location California

Type Hardware

Problem Description Unable to reboot the system

Severity

Status

Requester

ID wfault

First Name William

Last Name Faulkner

Email user1@us.oracle.com

Phone [0]

Resolution

Comment None

Resolved By None

History

Task Snapshot
 Future Participants
 Full task actions

#	Participant	Action	Action Date
1	Stage1		
1.1	jstein	Assigned	Apr 6, 2009 4:43 PM

Comments

Attachments

Any kind of change to the task details page, such as changing a priority or adding a comment or attachment, requires you to save the change before you go on to make any other changes.

The task details page has the following components:

- **Task Actions**—Lists the system actions that are possible for the task, such as **Request Information**, **Reassign**, **Renew**, **Suspend**, **Escalate**, and **Save**.
- **Action buttons**—Displays buttons for custom actions that are defined in the human task, such as setting task outcomes (for example, **Resolved** and **Unresolved** for a help desk request or **Approve** and **Reject** for a loan request). For the task initiator or a manager, **Withdraw** may also appear.

- **Details**—Displays task attributes, including the assignee, task creator, task number, state, priority, who acquired the task, and other flex fields. It also displays dates related to task creation, last update, and expiration date.
- **Contents**—Displays the payload. The fields displayed are specific to how the human task was created.
- **Requester**—Displays details (full name, contact information, and so on) about the task requester.
- **Resolution**—Displays any comments or resolution status.
- **History**—Displays the approval sequence and the update history for the task. See [Section 27.4.2, "Task History,"](#) for more information.
- **Comments**—Displays comments entered by various users who have participated in the workflow. A newly added comment and the commenter's user name are appended to the existing comments. A trail of comments is maintained throughout the life cycle of the task. To add or delete a comment, you must have permission to update the task.
- **Attachments**—Displays documents or reference URLs that are associated with a task. These are typically associated with the workflow as defined in the human task or attached and modified by any of the participants using the worklist. To add or delete an attachment, you must have permission to update the task. When adding file attachments, you can use an absolute path name or browse for a file.

Comments and attachments are shared between tasks and subtasks. Therefore, when you create a ToDo task and add comments and attachments, subtasks of this ToDo task include the same comments and attachments.

A user can view a task when associated with the task as the current assignee (directly or by group membership), the current assignee's manager, the creator, the owner, or a previous actor.

A user's profile determines his group memberships and roles. The roles determine a user's privileges. Apart from the privileges, the exact set of actions a user can perform is also determined by the state of the task, the custom actions, and restricted actions defined for the task flow at design time.

The following algorithm is used to determine the actions a user can perform on a task:

1. Get the list of actions a user can perform based on the privileges granted to him.
2. Get the list of actions that can be performed in the current state of the task.
3. Create a combined list of actions that appear on the preceding lists.
4. Remove any action on the combined list that is specified as a restricted action on the task.

The resulting list of actions is displayed in the task list page and the task details page for the user. When a user requests a specific action, such as claim, suspend, or reassign, the workflow service ensures that the requested action is contained in the list determined by the preceding algorithm.

Step 2 in the preceding algorithm deals with many cases. If a task is in a final, completed state (after all approvals in a sequential flow), an expired state, a withdrawn state, or an errored state, then no further update actions are permitted. In any of the these states, the task, task history, and subtasks (parent task in parallel flow) can be viewed. If a task is suspended, then it can only be resumed or withdrawn. A task that is assigned to a group must be claimed before any actions can be performed on it.

Note: If you act on a task from the task details page, for example, if you approve a task, then any unchanged task details data is saved along with the saved changes to the task. However if you act on a task from the Actions menu, then unchanged task details are not saved.

27.4.1 System Actions

The action bar displays system actions, which are available on all tasks based on the user's privileges. [Table 27-6](#) lists system actions.

Table 27-6 System Task Actions

Action	Description
Claim	If a task is assigned to a group or multiple users, then the task must be claimed first. Claim is the only action available in the Task Action list for group or multiuser assignments. After a task is claimed, all applicable actions are listed.
Escalate	If you are not able to complete a task, you can escalate it and add an optional comment in the Comments area. The task is reassigned to your manager (up one level in a hierarchy).
Pushback	Use this action to send a task down one level in the workflow to the previous assignee.
Reassign	If you are a manager, you can delegate a task to reportees. A user with <code>BPMWorkflowReassign</code> privileges can delegate a task to anyone.
Release	If a task is assigned to a group or multiple users, it can be released if the user who claimed the task cannot complete the task. Any of the other assignees can claim and complete the task.
Renew	If a task is about to expire, you can renew it and add an optional comment in the Comments area. The task expiration date is extended one week. A renewal appears in the task history. The renewal duration for a task can be controlled by an optional parameter. The default value is <code>P7D</code> (seven days).
Submit Information and Request Information	Use these actions if another user requests that you supply more information or to request more information from the task creator or any of the previous assignees. If reapproval is not required, then the task is assigned to the next approver or the next step in the business process.
Suspend and Resume	If a task is not relevant, you can suspend it. These options are available only to users who have been granted the <code>BPMWorkflowSuspend</code> role. Other users can access the task by selecting Previous in the task filter or by looking up tasks in the Suspended status. A suspension is indefinite. It does not expire until Resume is used to resume working on the task.
Withdraw	If you are the creator of a task and do not want to continue with it, for example, you want to cancel a vacation request, you can withdraw it and add an optional comment in the Comments area. The business process determines what happens next. You can use the Withdraw action on the home page by using the Creator task filter.

27.4.2 Task History

The task history maintains an audit trail of the actions performed by the participants in the workflow and a snapshot of the task payload and attachments at various points in the workflow. The short history for a task lists all versions created by the following tasks:

- Initiate task
- Reinitiate task
- Update outcome of task

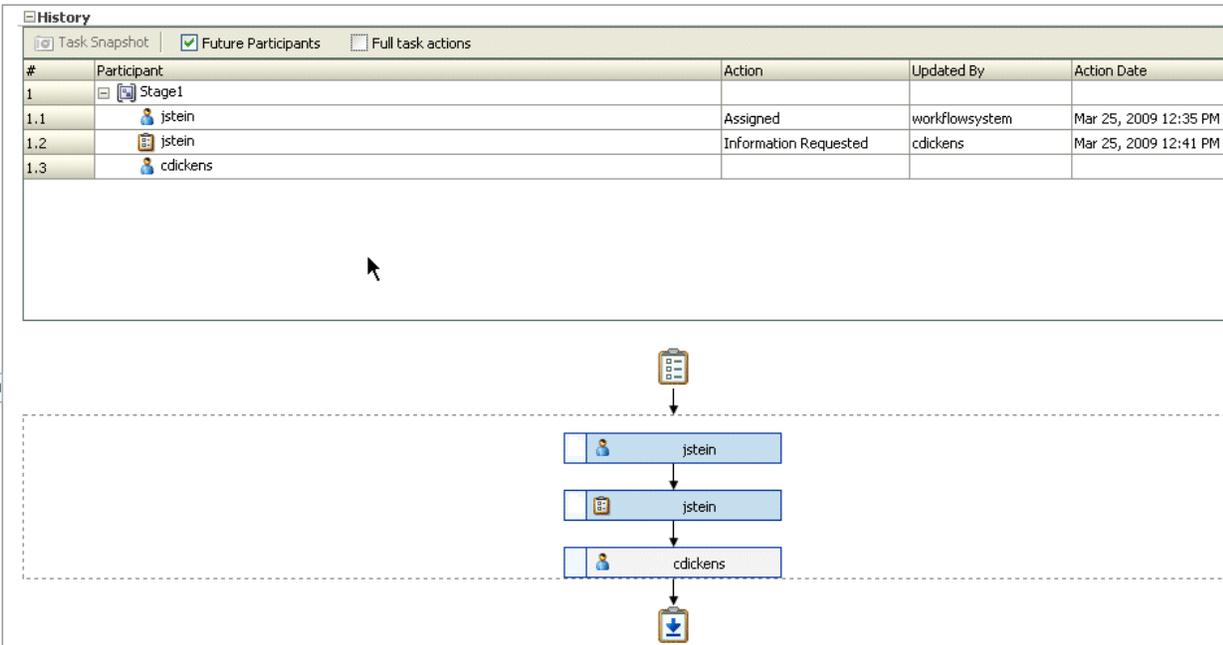
- Completion of task
- Erroring of task
- Expiration of task
- Withdrawal of task
- Alerting of task to the error assignee

You can include the following actions in the short history list by modifying the `shortHistoryActions` element.

- Acquire
- Ad hoc route
- Auto release of task
- Delegate
- Escalate
- Information request on task
- Information submit for task
- Override routing slip
- Update outcome and route
- Push back
- Reassign
- Release
- Renew
- Resume
- Skip current assignment
- Suspend
- Update

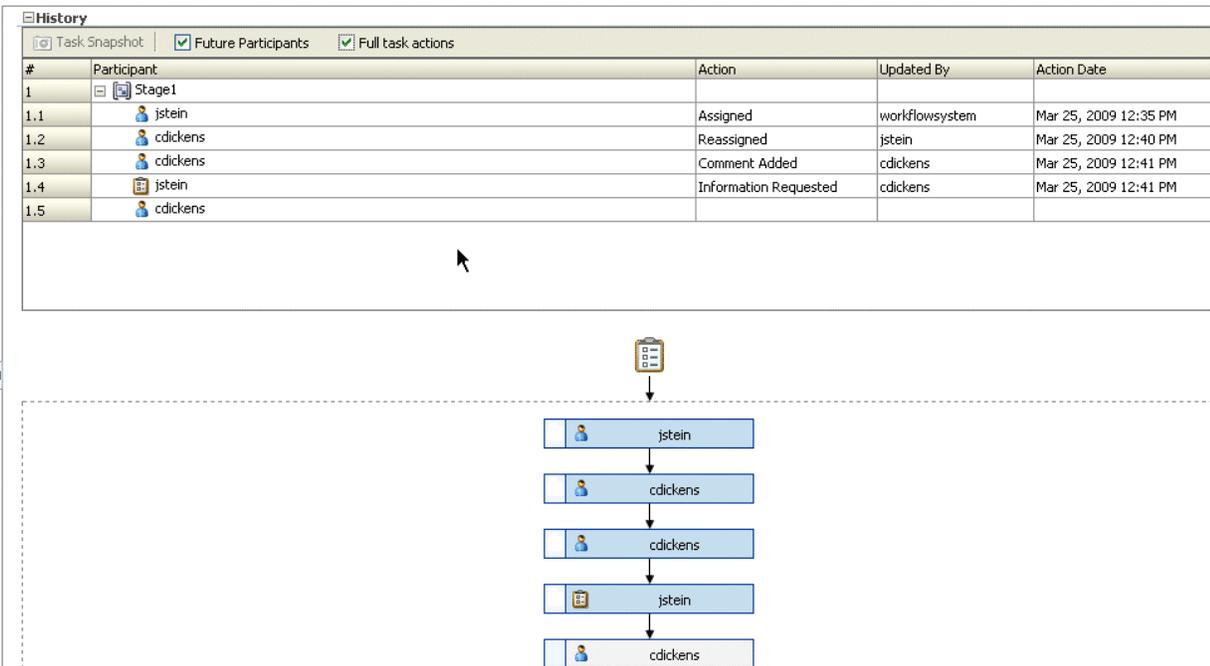
The history provides a graphical view of a task flow, as shown in [Figure 27-19](#).

Figure 27–19 History: Graphical View



Check **Full task actions** to see all actions performed, including those that do not make changes to the task, such as adding comments, as shown in [Figure 27–20](#).

Figure 27–20 History: Full Task Actions



Available ways to view the task history include:

- Take a task snapshot
- See future approvers
- See complete task actions

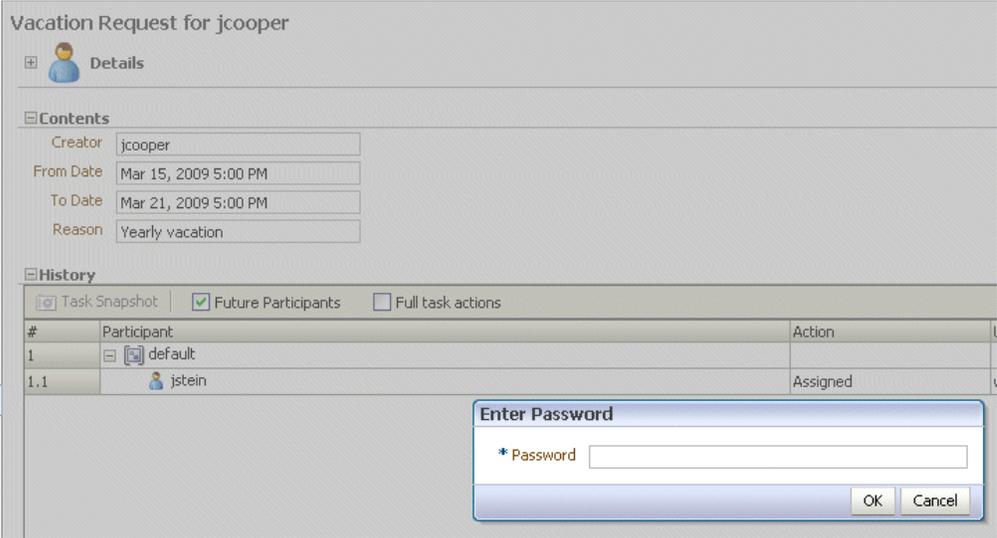
27.4.3 How To Act on Tasks

If the human task was designed to permit ad hoc routing, or if no predetermined sequence of approvers was defined, then the task can be routed in an ad hoc fashion in the worklist. For such tasks, a **Route** button appears on the task details page. From the Route page, you can look up one or more users for routing. When you specify multiple assignees, you can select whether the list of assignees is for simple (group assignment to all users), sequential, or parallel assignment.

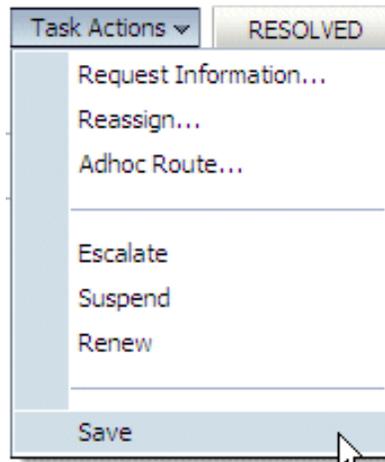
Parallel tasks are created when a parallel flow pattern is specified for scenarios such as voting. In this pattern, the parallel tasks have a common parent. The parent task is visible to a user only if the user is an assignee or an owner or creator of the task. The parallel tasks themselves (referred to as subtasks) are visible to whomever the task is assigned, just like any other task. It is possible to view the subtasks from a parent task. In such a scenario, the task details page of the parent task contains a **View SubTasks** button. The SubTasks page lists the corresponding parallel tasks. In a voting scenario, if any of the assignees updates the payload or comments or attachments, the changes are visible only to the assignee of that task. A user who can view the parent task (such as the final reviewer of a parallel flow pattern), can drill down to the subtasks and view the updates made to the subtasks by the participants in the parallel flow. In the worklist, you provide the percentage of votes required for approval.

If a human task was set up to require a password, then when you act on it, you must provide the password, as shown in [Figure 27-21](#).

Figure 27-21 Acting on a Task That Requires a Password



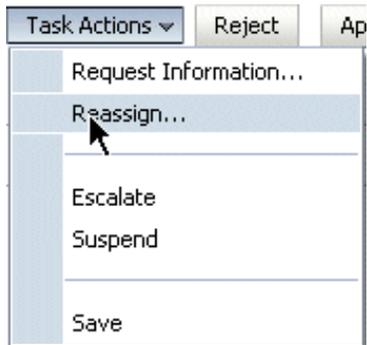
Note: Any kind of change to the task details page, such as changing a priority or adding a comment or attachment, requires you to save the change.



To reassign or delegate a task:

1. From the **Task Actions** list, select **Reassign**, as shown in [Figure 27–22](#).

Figure 27–22 *Reassigning a Task*

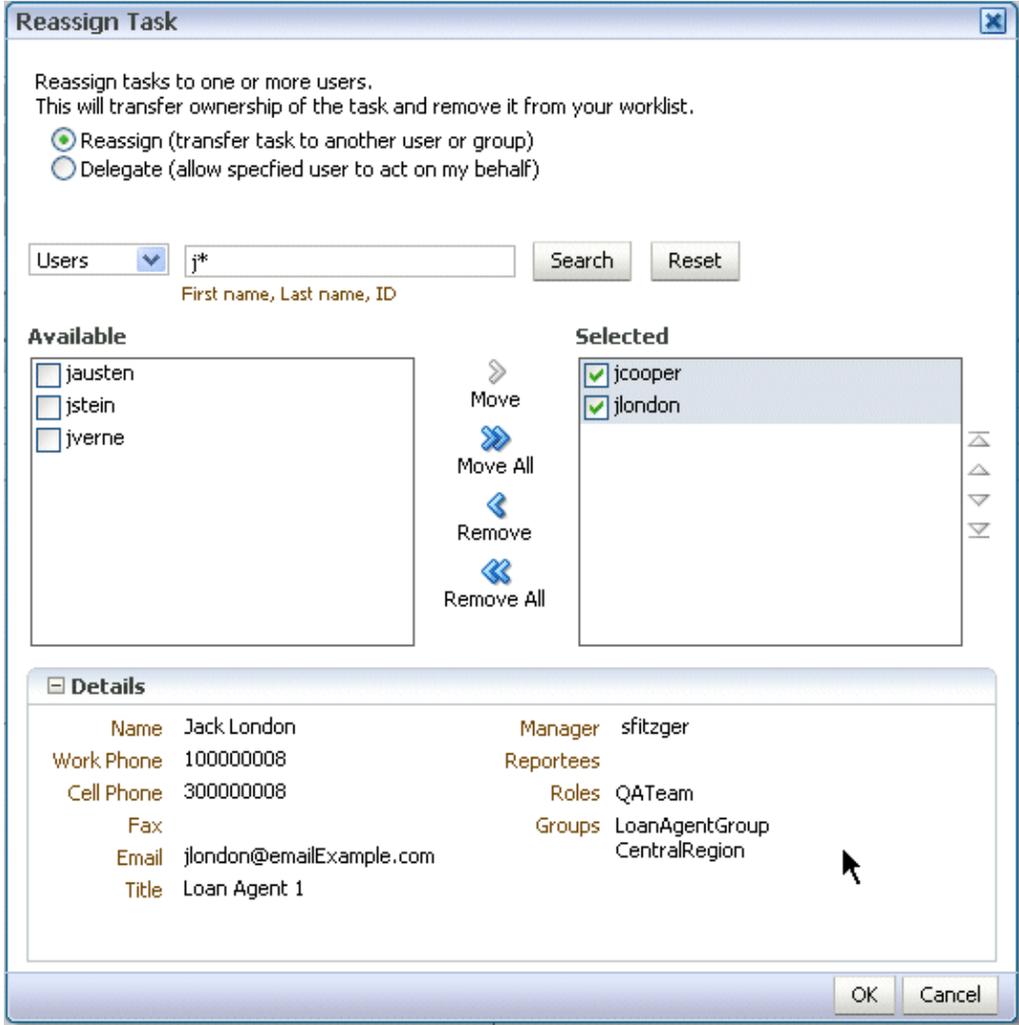


2. Select **Reassign** or **Delegate**.

Delegate differs from **Reassign** in that the privileges of the delegatee are based on the delegator's privileges. This function can be used by managers' assistants, for example.

3. Provide or browse for a user or group name, as shown in [Figure 27–23](#).

Figure 27-23 Reassigning a Task



A supervisor can always reassign tasks to any of his reportees. Users with the BPMWorkflowReassign role can assign tasks to any users in the organization.

- 4. Move names to the **Selected** area and click **OK**.

You can reassign to multiple users or groups. One of the assignees must claim the task, as shown in Figure 27-24.

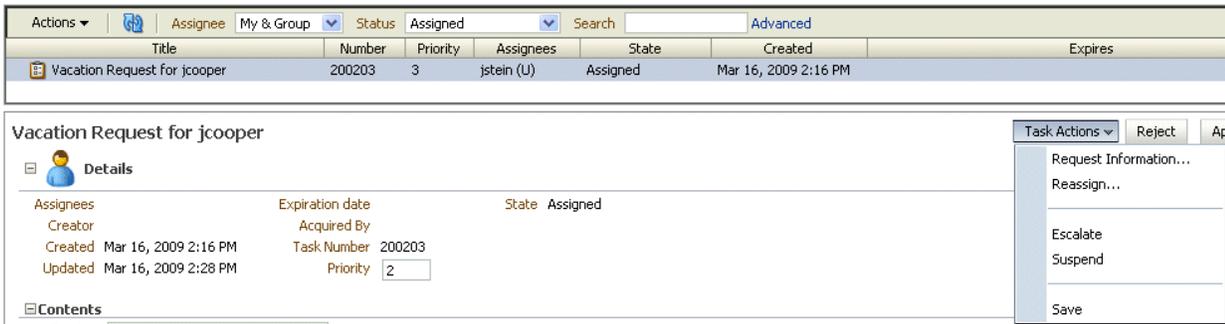
Figure 27-24 Claiming a Task



To request information:

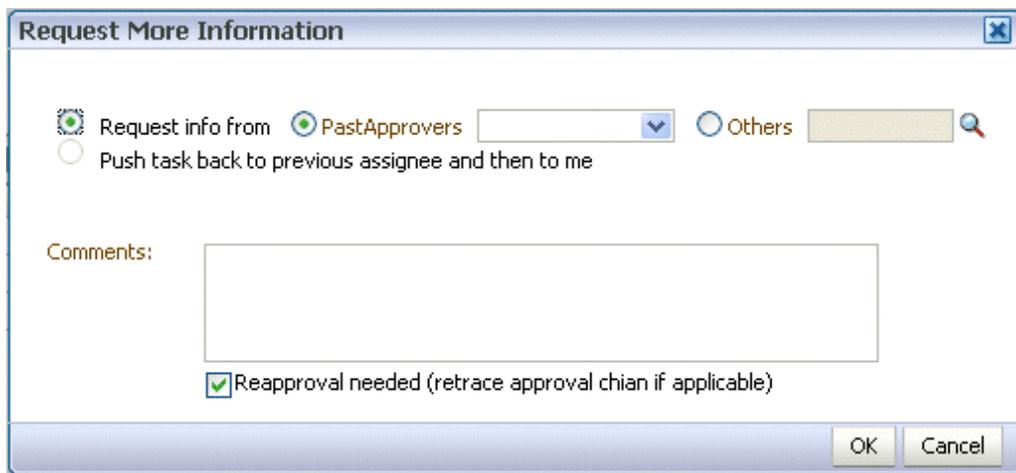
1. From the **Task Actions** list, select **Request Information**, as shown in [Figure 27–25](#).

Figure 27–25 Requesting Information



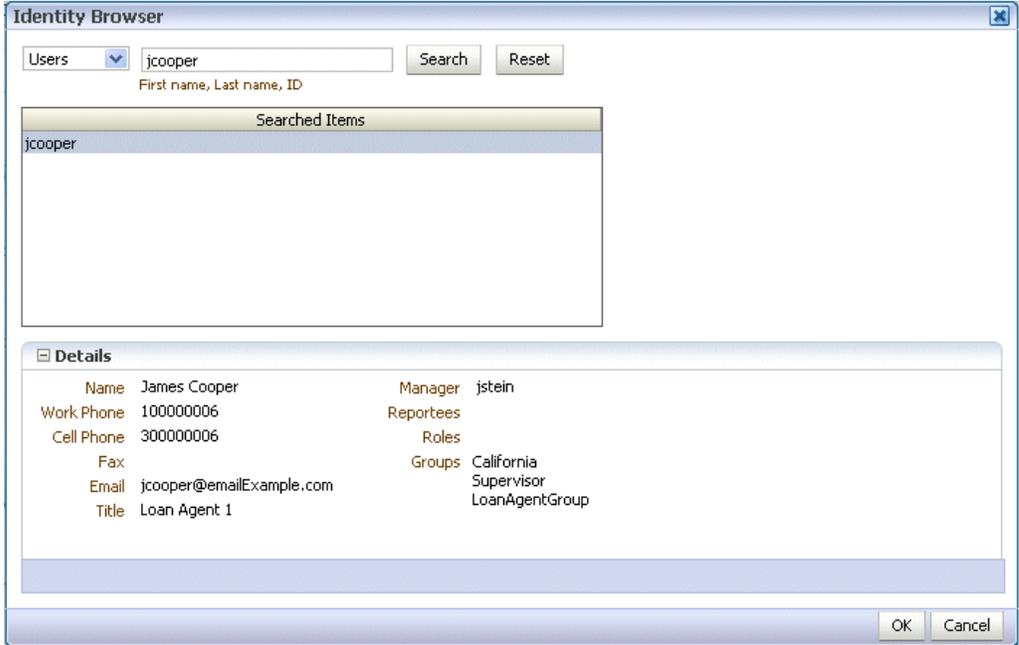
2. Request information from a past approver or search for a user name, or push the task back to the previous assignee, as shown in [Figure 27–26](#).

Figure 27–26 Requesting Information from Past Approvers or Another User, or Pushing the Task Back



If you use the **Search** icon to find a user name, the Identity Browser appears, as shown in [Figure 27–27](#).

Figure 27-27 Identity Browser

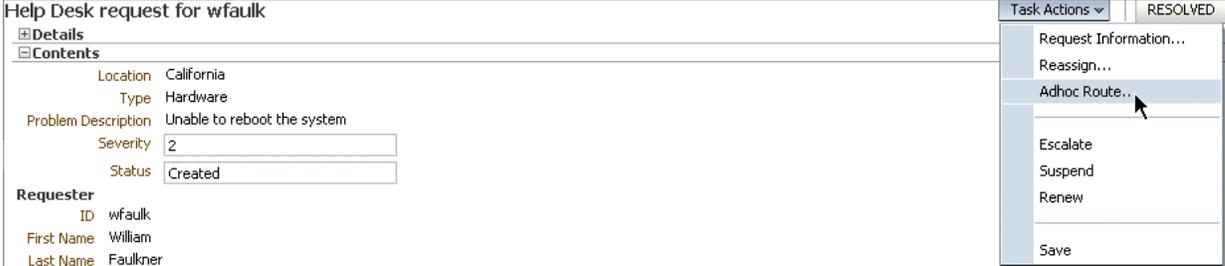


3. Click OK.

To route a task:

1. From the Task Actions list, select Adhoc Route, as shown in Figure 27-28.

Figure 27-28 Ad Hoc Routing



2. Select an action and a routing option, as shown in Figure 27-29.

Figure 27–29 Routing a Task

Route Task

RESOLVED and route to: Comments:

Single Approver
 Group Vote
 Chain of Single Approvers

Groups Search Reset

First name, Last name, ID

Available Selected

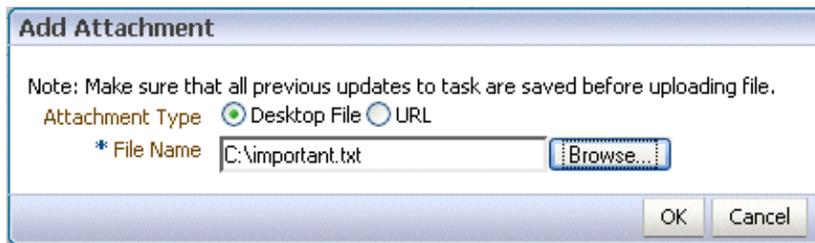
Move
 Move All
 Remove
 Remove All

Details

Please select an item to see its details

OK Cancel

- **Single Approver:** Use this option for a single user to act on a task. If the task is assigned to a role or group with multiple users, one of the members must claim the task and act on it.
- **Group Vote:** Use this option when multiple users, working in parallel, must act, such as in a hiring situation when multiple users vote to hire or reject an applicant. You specify the voting percentage that is needed for the outcome to take effect, such as a majority vote or a unanimous vote, as shown in [Figure 27–30](#).

Figure 27–32 Adding a Worklist Attachment

Note: Attachment file names that use a multibyte character set (MBCS) are not supported.

Attachments of up to 2 MB can be uploaded. You can modify this setting by setting the context parameter in `web.xml` as follows:

```
<context-param>
  <param-name>oracle.adf.view.faces.UPLOAD_MAX_DISK_
SPACE</param-name>
  <param-value>1024000</param-value>
</context-param>
```

4. From the Task Actions list, click **Save**.

27.4.4 How To Act on Tasks That Require a Digital Signature

The worklist supports the signature policy created in the human task:

- **No signature required** — Participants can send and act on tasks without providing a signature.
- **Password required** — Participants must specify their login passwords.
- **Digital certificate (signature) required** — Participants must possess a digital certificate before being able to send and act on tasks. A digital certificate contains the digital signature of the certificate-issuing authority so that anyone can verify that the certificate is real. A digital certificate establishes the participant's credentials. It is issued by a certification authority (CA). It contains your name, a serial number, expiration dates, a copy of the certificate holder's public key (used for encrypting messages and digital signatures), and the digital signature of the certificate-issuing authority so that a recipient can verify that the certificate is real.

When you act on a task that has a signature policy, the **Sign** button appears, as shown in [Figure 27–33](#).

Figure 27-33 Digital Signature Task Details

Digital Signature Task Details Sign

[Back To History](#)

Approve Order

Task Number: 200002	Creator: jstein	Assignees: jcooper
State: Assigned	Created Date: 5/15/09 6:56:47 AM	Acquired By:
Outcome	Updated Date: 5/15/09 6:56:47 AM	
Priority: <input type="text" value="3"/>	Expiration Date:	

Customer Information

CustID: 10
 ID: 23
 Street:
 City:
 State:
 Zip:
 Country:
 Next

The evidence store service is used for digital signature storage and nonrepudiation of digitally signed human tasks. You can search the evidence store, as shown in Figure 27-34.

Figure 27-34 The Evidence Store

[My Tasks](#) | [Initiated Tasks](#) | [Administration Tasks](#) | [Administration](#) | **Evidence Search**

Search Evidence Store

Match All Any

Signer: Signed After: Signed Before:

Advanced

Signature Policy: Verified Date: Task Outcome:

Status: Task Number:

[Validate](#)

Signature Policy	Signer	Status	Creation Date	Signed Date

See [Section 29.1.10, "Evidence Store Service and Digital Signatures"](#) for more information.

To provide a digital signature:

1. In the upper right corner of Oracle BPM Worklist, click **Preferences**.
2. In the navigation bar on the left, click **Certificates**.
3. Upload the certificate to use to sign your decision, as shown in [Figure 27–35](#).

Figure 27–35 Uploading a Certificate

Note the following important points when providing your certificate to the system. Otherwise, you cannot use your certificate to sign your decisions on tasks.

- The PKCS7 file format is a binary certificate format. Select this option if you have a standalone certificate file stored on your disk.
 - The PKCS12 file format is a keystore format. Select this option if you have your certificate stored inside a keystore.
 - If you want to copy and paste the contents of the certificate, select **Type or Paste Certificate Contents** and paste the BASE64-encoded text into the field. Do not paste a certificate in any other format into this field. Likewise, if you choose to upload a certificate, do not try to upload a BASE64-encoded certificate. Only PKCS12 and PKCS7 formatted files are supported for uploads.
4. Return to the task list by clicking the **Home** link in the upper-right corner of Oracle BPM Worklist.
 5. Click a task that you want to either approve or reject.
The task details are displayed.
 6. Click either **Approve** or **Reject**.
Details about the digital signature are displayed.
 7. For a task that has a signature policy, click **Sign**.

The Text Signing Report dialog appears.

8. Select the certificate from the dropdown list to use to sign your decision.
9. Enter the master password of the web browser that you are using.
10. Click OK.

The web browser signs the string displayed in the upper half of the Text Signing Request with the certificate you selected and invokes the action (approval or rejection) that you selected. The task status is appropriately updated in the human workflow service.

For more information about how certificates are uploaded and used, see [Section 29.1.10, "Evidence Store Service and Digital Signatures."](#)

27.5 Approving Tasks

Table 27-7 describes the type of actions that can be performed on tasks by the various task approvers.

Table 27-7 Task Actions and Approvers

Task Action	Admin	Owner (+ Owner Group)	Assignee (+ Assignee Manager + Assignee Group + Proxy Assignee)	Creator	Reviewer	Approver
Acquire (Claim)	No	Yes	Yes	No	No	No
Custom	No	Yes ¹	Yes ¹	No	No	No
Delegate	No	No	Yes	No	No	No
Delete	Yes ²	Yes ²	Yes ²	Yes ²	No	No
Error	No	No	Yes ³	No	No	No
Escalate	Yes ⁴	Yes ⁴	Yes	No	No	No
Info Request	No	No	Yes	No	No	No
Info Submit	No	No	Yes	No	No	No
Override Routing Slip	Yes	Yes	No	No	No	No
Push Back	No	No	Yes	No	No	No
Purge	Yes ²	No	No	No	No	No
Reassign	Yes ⁵	Yes ⁵	Yes (No for proxy assignee)	No	No	No
Release	Yes	Yes	Yes	No	No	No
Renew	No	Yes	Yes	No	No	No
Resume	Yes	Yes	Yes	No	No	No
Route	No	Yes	Yes	No	No	No
Skip Current Assignment	Yes	Yes	No	No	No	No
Suspend	Yes	Yes	Yes	No	No	No

Table 27-7 (Cont.) Task Actions and Approvers

Task Action	Admin	Owner (+ Owner Group)	Assignee (+ Assignee Manager + Assignee Group + Proxy Assignee)	Creator	Reviewer	Approver
Update	No	Yes	Yes	Yes	No	No
Update Attachment	No	Yes	Yes	Yes	Yes	No
Update Comment	No	Yes	Yes	Yes	Yes	No
View Process History	Yes	Yes	Yes	Yes	No	No
View Sub Tasks	Yes	Yes	Yes	No	No	No
View Task History	Yes	Yes	Yes	Yes	Yes	Yes
Withdraw	No	Yes	No	Yes	No	No

¹ Not valid for ToDo tasks

² Valid only for ToDo tasks

³ Applicable for tasks in alerted states

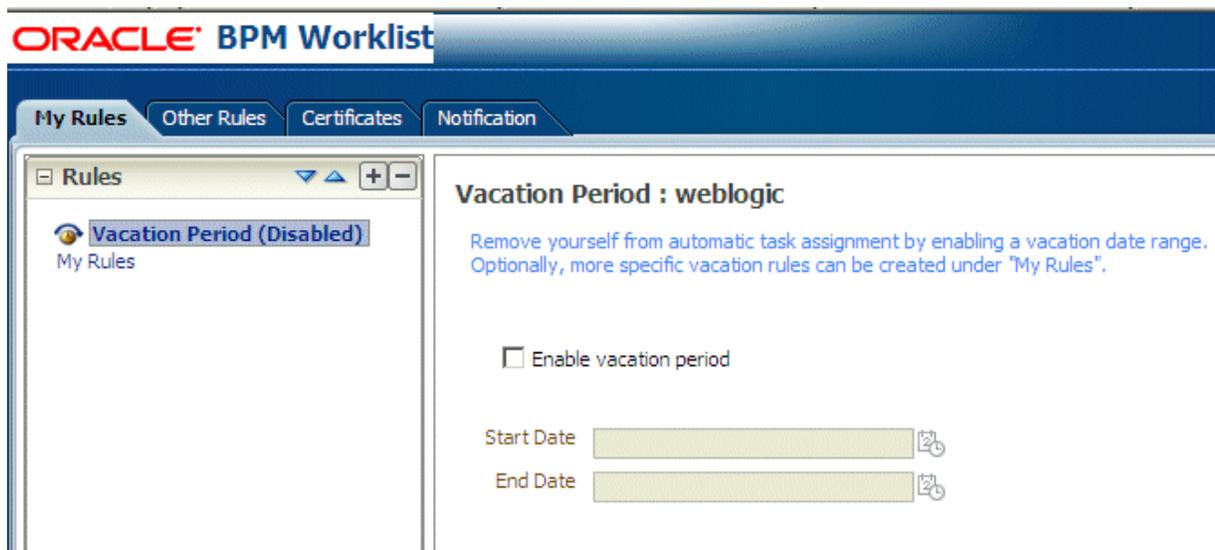
⁴ Without claim and escalate to current assignee's manager

⁵ Without claim

27.6 Setting a Vacation Period

You can set a vacation period so that you are removed from automatic task assignment during the dates you specify, as shown in [Figure 27-36](#).

Figure 27-36 Setting a Vacation Period



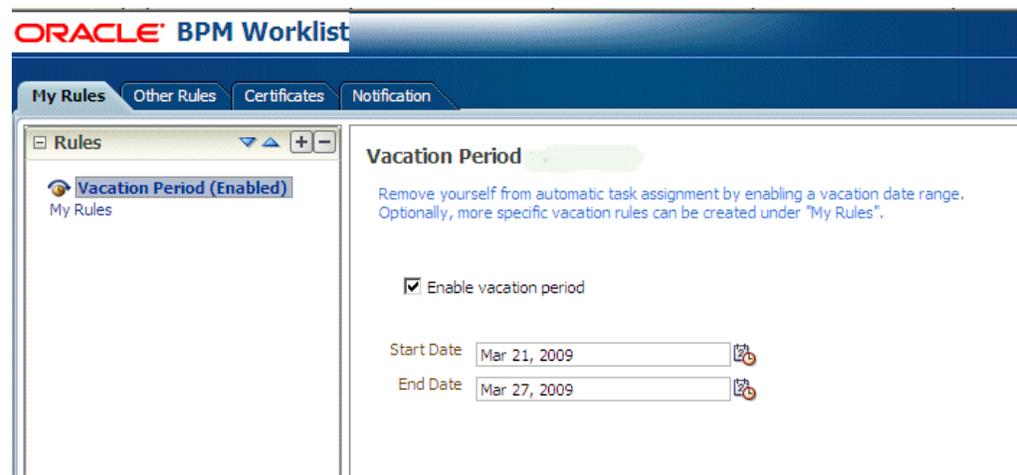
Vacation rules are not executed for ToDo tasks. See [Section 27.7, "Setting Rules,"](#) for how to set a vacation rule that is synchronized with the vacation period.

To create a vacation period:

1. Click the **Preferences** link.
The **My Rules** tab is displayed.
2. Click **Enable vacation period**.
3. Provide start and end dates.
4. Click **Save**.

The vacation period is enabled, as shown in [Figure 27–37](#).

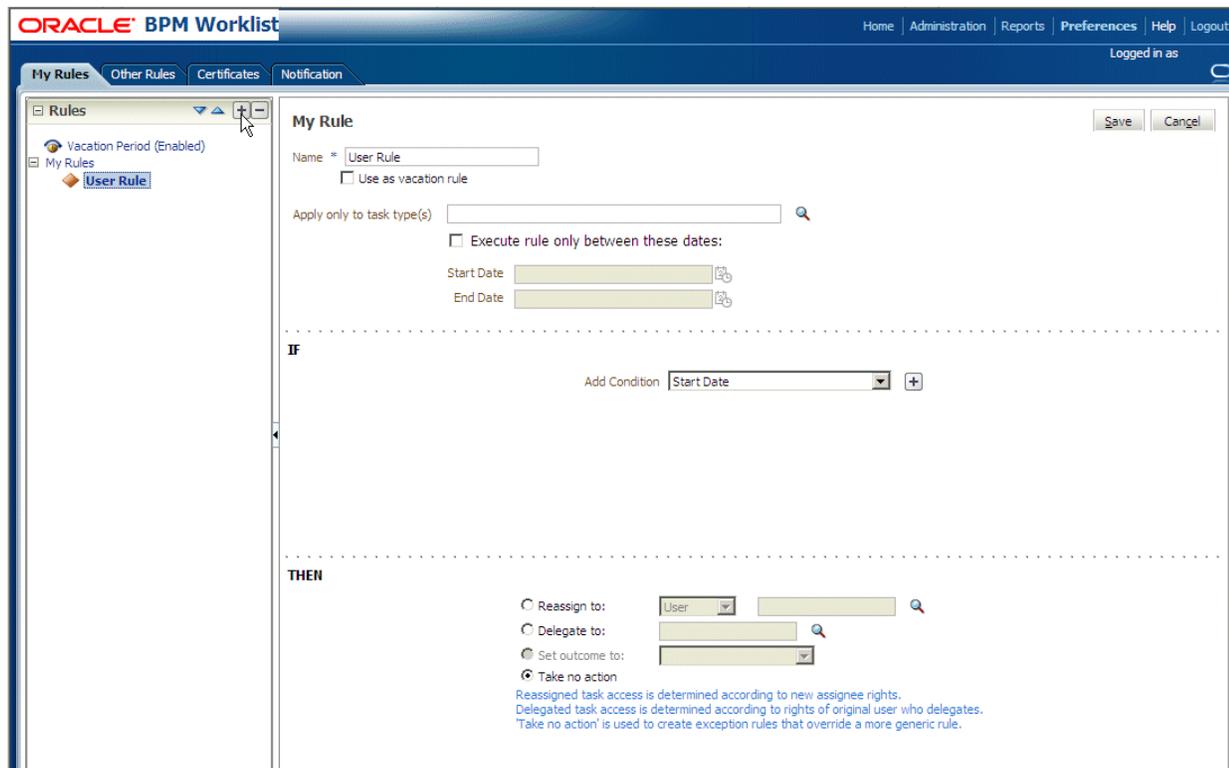
Figure 27–37 Enabling a Vacation Period



27.7 Setting Rules

Rules act on tasks, either a specific task type or all the tasks assigned to a user or group. [Figure 27–38](#) shows where you set rules, including vacation rules (different from the vacation period settings described in [Section 27.6](#), "Setting a Vacation Period").

Figure 27–38 Creating a Rule



A rule cannot always apply in all circumstances in which it is used. For example, if a rule applies to multiple task types, it may not be possible to set the outcome for all tasks, since different tasks can have different outcomes.

Rules are executed in the order in which they are listed. Rules can be reordered by using the up and down buttons in the header, as shown in [Figure 27–38](#).

If a rule meets its filter conditions, then it is executed and no other rules are evaluated. For your rule to execute, you must be the only user assigned to that task. If the task is assigned to multiple users (including you), the rule does not execute.

You cannot specify business rules for ToDo tasks

27.7.1 How To Create User Rules

Specify the following when creating a user rule:

- Rule name
- If the rule is a vacation rule. See [Section 27.6, "Setting a Vacation Period,"](#) for how to set the vacation period that is synchronized with the vacation rule.
- Which task or task type the rule applies to—If unspecified, then the rule applies to all tasks. If a task type is specified, then any flex field attributes mapped for that task type can be used in the rule condition.
- When the rule applies
- Conditions on the rule—These are filters that further define the rule, such as specifying that a rule acts on priority 1 tasks only, or that a rule acts on tasks created by a specific user. The conditions can be based on standard task attributes

and any flex fields that have been mapped for the specific tasks. See [Section 27.10.1, "How To Map Flex Fields,"](#) for more information.

User rules do the following actions:

- **Reassign to**—You can reassign tasks to subordinates or groups you manage. If you have been granted the BPMWorkflowReassign role, then you can reassign tasks to any user or group.
- **Delegate to**—You can delegate to any user or group. Any access rights or privileges for completing the task are determined according to the original user who delegated the task. (Any subsequent delegations or re-assignments do not change this from the original delegating user.)
- **Set outcome to**—You can specify an automatic outcome if the workflow task was designed for those outcomes, for example, accepting or rejecting the task. The rule must be for a specific task type. If a rule is for all task types, then this option is not displayed.
- **Take no action**—Use this action to prevent other more general rules from applying. For example, to reassign all your tasks to another user while you are on vacation, with the exception of loan requests, for which you want no action taken, then create two rules. The first rule specifies that no action is taken for loan requests; the second rule specifies that all tasks are reassigned to another user. The first rule prevents reassignment for loan requests.

To create a user rule:

1. Click the **Preferences** link
The **My Rules** tab is displayed.
2. In the **Rules** area, click **My Rules** and click **Add**.
3. In the **My Rule** area, do the following and click **Save**:
 - Provide a name for the rule.
 - Select **Use as a vacation rule** if you are creating a vacation rule. The start and end dates of the rule are automatically synchronized with the vacation period.
 - Browse for task types to which the rule applies.
 - Select **Execute rule only between these dates** and provide rule execution dates.
 - In the **IF** area, add rule conditions.
 - In the **THEN** area, select actions to be taken: **Reassign to**, **Delegate to**, **Set outcome to**, or **Take no action**), as shown in [Figure 27-38](#).

The new rule appears under the **My Rules** node.

27.7.2 How To Create Group Rules

Creating a group rule is similar to creating a user rule, with the addition of a list of the groups that you (as the logged-in user) manage. Examples of group rules include:

- Assigning tasks from a particular customer to a member of the group
- Ensuring an even distribution of task assignments to members of a group by using round-robin assignment
- Ensuring that high-priority tasks are routed to the least busy member of a group

Group rules do the following actions:

- **Assign to member via**—You can specify a criterion to determine which member of the group gets the assignment. This dynamic assignment criterion can include round-robin assignment, assignment to the least busy group member, or assignment to the most productive group member. You can also add your custom functions for allocating tasks to users in a group.
- **Assign to**—As with user rules, you can assign tasks to subordinates or groups you directly manage. If you have been granted the BPMWorkflowReassign role, then you can reassign tasks to any user or group (outside your management hierarchy).
- **Take no action**—As with user rules, you can create a rule with a condition that prevents a more generic rule from being executed.

To create a group rule:

1. Click the **Preferences** link
2. Click the **Other Rules** tab.
3. Select **Group** from the list.
4. Enter a group name and click the **Search** icon, or enter a group name.

The Identity Browser opens for you to find and select a group.

5. Select the group name under the **Group Rules** node and click **Add**, as shown in [Figure 27–39](#).

Figure 27–39 *Creating a Group Rule*

The screenshot displays the 'Group Rules' configuration interface. On the left, a search bar contains 'weblogic' and a 'Go' button. Below it, a tree view shows 'Group Rules' expanded to show 'weblogic' and 'Group Rule'. The main area is titled 'Group Rules' and contains the following fields and options:

- Name:** Group Rule
- Apply only to task type(s):** (empty search field)
- Execute rule only between these dates:
 - Start Date: (calendar icon)
 - End Date: (calendar icon)
- IF:**
 - Add Condition: Start Date (dropdown menu) + (+)
- THEN:**
 - Assign to member via: Least Busy (dropdown menu)
 - Assign to: User (dropdown menu) + (search icon)
 - Take no action

Below the 'THEN' section, there is a note: "Reassigned task access is determined according to new assignee rights. Delegated task access is determined according to rights of original user who delegates." Buttons for 'Save' and 'Cancel' are located at the top right of the main configuration area.

6. Provide group rule information and click **Save**.
 - Provide a name for the rule.
 - Browse for task types to which the rule applies.
 - Provide rule execution dates.
 - In the **IF** area, add rule conditions.
 - In the **THEN** area, select the actions to be taken (or none) (**Assign to member via**, **Assign to**, or **Take no action**), as shown in [Figure 27–39](#).

The new rule appears under the **Group Rules** node.

27.7.3 Assignment Rules for Tasks with Multiple Assignees

If a task has more than one assignee, then assignment rules are not evaluated for the task, and the task is not automatically routed. This is because each of the task's assignees can define assignment rules, which can potentially provide conflicting actions to take on the task. Only tasks that are assigned exclusively to a single user are routed by the assignment rules.

For example, consider the following sequence:

1. A rule is created for user cdickens to reassign all assigned requests to user jstein.
2. User jcooper reassigns the allocated tasks to cdickens and cdoyle.
3. Cdickens claims the task, and the task appears in their inbox.

The task is not automatically reassigned to jstein. The task is routed to jstein, following the assignment rule set for cdickens, if user jcooper explicitly re-assigns the task only to cdickens instead of reassigning the task to multiple users (cdickens and cdoyle).

27.8 Using the Worklist Administration Functions

Administrators are users who have been granted the BPMWorkflowAdmin role. Administration functions include the following:

- Managing other users' or groups' rules
- Setting the worklist display (application preferences)
- Mapping flex fields

An administrator can view and update all tasks assigned to all users. An administrator's **Assignee** filter displays **Admin** when the Admin tab is selected.

27.8.1 How To Manage Other Users' or Groups' Rules (as an Administrator)

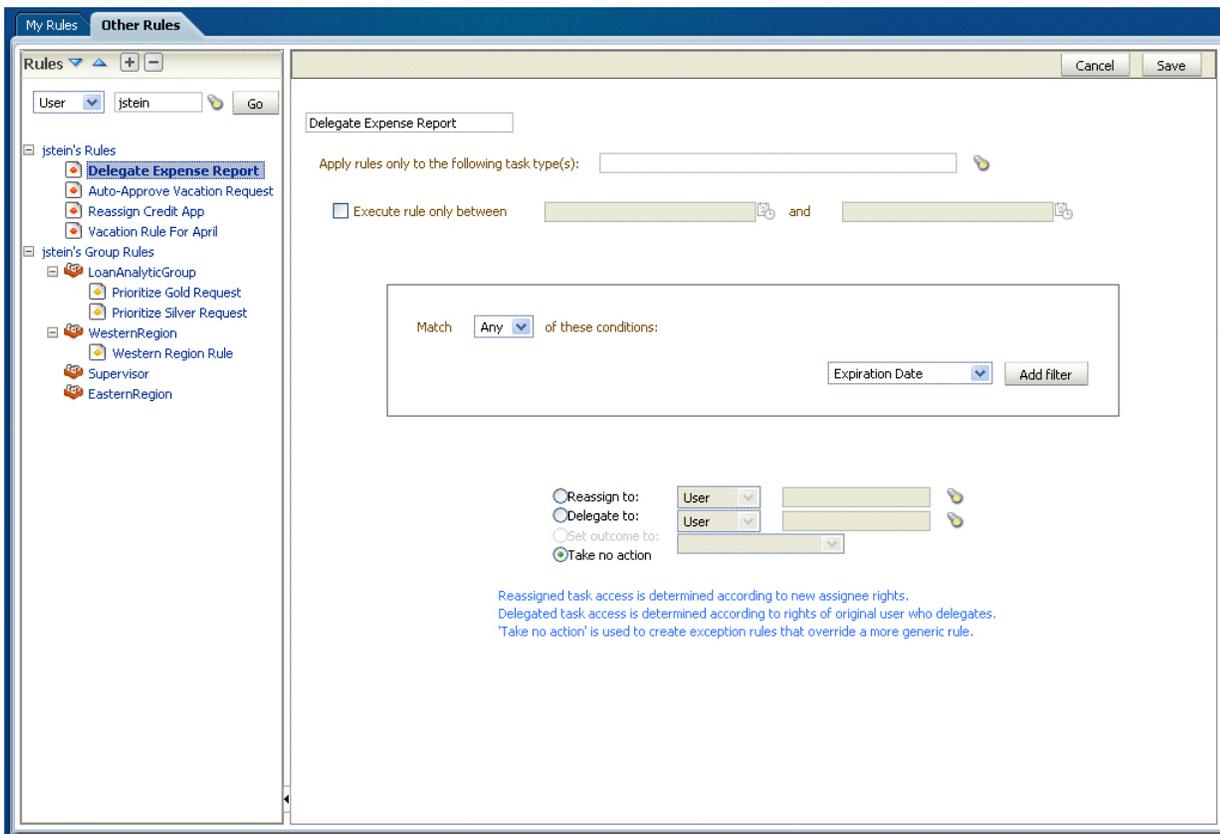
This function is useful for fixing a problem with a rule. Also, for a user who no longer works for the company, administrators can set up a rule for that user so that all tasks assigned to the user are automatically assigned to another user or group.

To create a rule for another user or group:

1. From the task list page, click the **Rules** link.
2. Click the **Other Rules** tab.
3. Search for the user or group for whom rules are to be created, as shown in [Figure 27–40](#).

Figure 27–40 *Creating Rules for Another User or Group*

4. Click a user rules node, or click a group name (for a group rule).
5. Click the **Add** icon to create a rule.
6. Provide rule information, as shown in [Figure 27–41](#), and click **Save**.

Figure 27–41 *Defining Rules for Another User or Group*

27.8.2 How To Set the Worklist Display (Application Preferences)

Application preferences customize the appearance of the worklist. Administrators can specify the following:

- **Login page realm label**—If the identity service is configured with multiple realms, then the Oracle BPM Worklist login page displays a list of realm names. LABEL_ LOGIN_REALM specifies the resource bundle key used to look up the label to

display these realms. The term *realm* can be changed to fit the user community—terms such as *country*, *company*, *division*, or *department* may be more appropriate. Administrators can customize the resource bundle, specify a resource key for this string, and then set this parameter to point to the resource key.

- **Global branding icon**—This is the image displayed in the top left corner of every page of the worklist. (The Oracle logo is the default.) Administrators can provide a .gif, .png, or .jpg file for the logo. This file must be in the `public_html` directory.
- **Resource bundle**—An application resource bundle provides the strings displayed in the worklist. By default, this is the class at:

```
oracle.bpel.worklistapp.resource.WorklistResourceBundle
```

Administrators can change the strings shown in the application by copying `WorkflowResourceBundle` and creating their own. This parameter allows administrators to specify the classpath to this custom resource bundle.

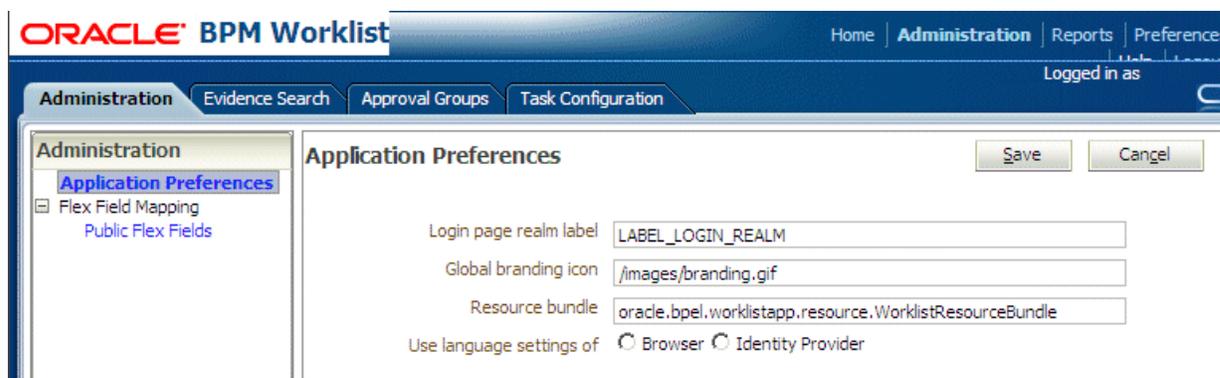
Administrators must extend `WorklistResourceBundle.java` by adding their resource strings. Administrators can change the strings shown in the application by copying `WorkflowResourceBundle` and creating their own. This parameter allows administrators to specify the classpath to this custom resource bundle. Then administrators create a JAR file from the compiled resource bundle and copy it under

```
SOA_Oracle_Home\j2ee\home\applications\worklist\worklist\WEB-INF\lib
```

- **Use language settings of**—Select the browser or the identity provider.
The Identity Provider that stores information on worklist users can store the user's locale, which can be used to determine the worklist display language. Alternatively, the user's browser can supply the locale information. This parameter determines which is used as the source for determining the worklist application display language.

To specify application preferences:

1. Click the **Administration** tab.
2. Click **Application Preferences**.
3. Browse for the locations of the application preferences (login page realm label, branding icon, or resource bundle).



4. Click **Save**.

27.9 Specifying Notification Settings

You can configure the notification settings to control how, when, and where you receive messages in cases when you have access to multiple communication channels (delivery types). Specifically, you can define messaging filters (delivery preferences) that specify the channel to which a message should be delivered, and under what circumstances.

For example, you might want to create filters for messages received from customers with different Service Level Agreements (SLA), specifying to be notified through business phone and SMS channels for customers with a premium SLA and by EMAIL for customers with a nonpremium SLA.

27.9.1 Messaging Filter Rules

A messaging filter rule consists of *rule conditions* and *rule actions*. A rule condition consists of a rule attribute, an operator, and an associated value. A rule action is the action to be taken if the specified conditions in a rule are true.

27.9.1.1 Data Types

Table 27–8 lists data types supported by messaging filters. Each attribute has an associated data type, and each data type has a set of predefined comparison operators.

Table 27–8 Data Types Supported by Messaging Filters

Data Type	Comparison Operators
Date	isEqual, isNotEqual, isGreaterThan, isGreaterThanOrEqual, isLessThan, isLessThanOrEqual, Between, isWeekday, isWeekend
Time	isEqual, isNotEqual, Between
Number	isEqual, isNotEqual, Between, isGreaterThan, isGreaterThanOrEqual, isLessThan, isLessThanOrEqual
String	isEqual, isNotEqual, Contains, NotContains

Note: The String data type does not support regular expressions.

27.9.1.2 Attributes

Table 27–9 lists the predefined attributes for messaging filters.

Table 27–9 Predefined Attributes for Messaging Filters

Attribute	Data Type
Total Cost	Number
From	String
Expense Type	String
To	String
Application Type	String
Duration	Number
Application	String
Process Type	String

Table 27–9 (Cont.) Predefined Attributes for Messaging Filters

Attribute	Data Type
Status	String
Subject	String
Customer Type	String
Time	Time
Group Name	String
Processing Time	Number
Date	Date
Due Date	Date
User	String
Source	String
Amount	Number
Role	String
Priority	String
Customer Name	String
Expiration Date	Date
Order Type	String
Organization	String
Classification	String
Service Request Type	String

27.9.2 Rule Actions

For a given rule, a messaging filter can define the following actions:

- **Send No Messages:** Do not send a message to any channel.
- **Send Messages to All Selected Channels:** Send a message to all specified channels in the address list.
- **Send to the First Available Channel:** Send a message serially to channels in the address list until one successful message is sent. This entails performing a send to the next channel when the current channel returns a failure status. This filter action is not supported for messages sent from the human workflow layer.

27.9.3 Managing Messaging Channels

In Oracle BPM Worklist, messaging channels represent both physical channels, such as business mobile phones, and also email client applications running on desktops. Specifically, Oracle BPM Worklist supports the following messaging channels:

- EMAIL
- IM
- MOBILE
- SMS
- VOICE

- WORKLIST

Note the following about message channels:

- Addresses for messaging channels are fetched from the configured identity store.
- SMS and MOBILE notifications are sent to the mobile phone number.
- VOICE notifications are sent to the business phone number.
- No special notification is sent when the messaging channel preference is WORKLIST. Instead, log in to Oracle BPM Worklist to view tasks.
- EMAIL is the default messaging channel preference when a preferred channel has not been selected.

You can use the **Messaging Channels** tab to view, create, edit, and delete messaging channels.

27.9.3.1 Viewing Your Messaging Channels

You can display your existing messaging channels.

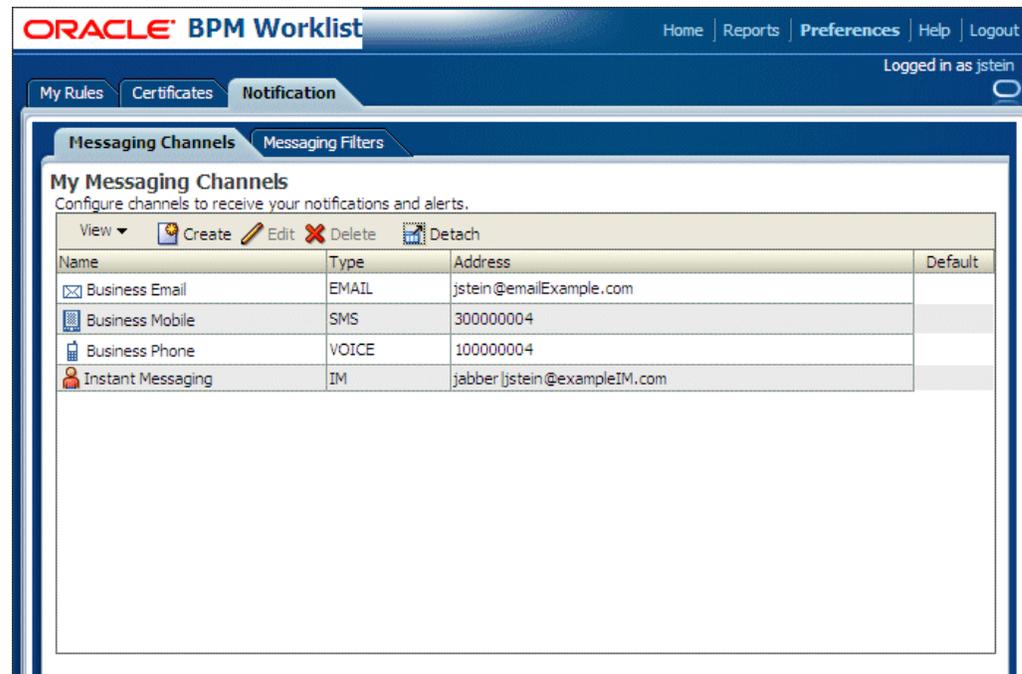
To view messaging channels:

1. Click the **Preferences** link.
2. Click the **Notification** tab.
3. Click the **Messaging Channels** tab.

The My Messaging Channels list appears ([Figure 27-42](#)) and displays the following information:

- **Name:** The name of the messaging channel.
- **Type:** The type of messaging channel, such as EMAIL or SMS.
- **Address:** The address for the channel, such as a phone number or email address.
- **Default:** Specifies whether this channel is the default messaging channel.

Figure 27-42 Messaging Channels



4. Click **View** > **Columns** and select the columns to display or hide.

You can also click **View** > **Reorder Columns** to display a dialog box to reorder the displayed columns.

Messaging channel names and addresses are retrieved from the underlying identity store, such as Oracle Internet Directory.

27.9.3.2 Creating, Editing, and Deleting a Messaging Channel

Oracle BPM Worklist uses an underlying identity store, such as Oracle Internet Directory, to manage messaging channels and addresses. Therefore, you cannot directly create, modify, or delete messaging channels using Oracle BPM Worklist.

To perform these actions, contact the system administrator responsible for managing your organization's identity store.

27.9.4 Managing Messaging Filters

You can use the **Messaging Filters** tab to define filters that specify the types of notifications you want to receive along with the channels through which to receive these notifications. You can do this through a combination of comparison operators (such as *is equal to*, *is not equal to*), attributes that describe the notification type, content, or source, and notification actions, which send the notifications to the first available messaging channels, all messaging channels, or to no channels (effectively blocking the notification).

For example, you can create a messaging filter called *Messages from Lise*, that retrieves all messages addressed to you from your boss, Lise. Notifications that match all of the filter conditions might first be directed to your business mobile phone, for instance, and then to your business email if the first messaging channel is unavailable.

27.9.4.1 Viewing Messaging Filters

You can display your existing messaging filters.

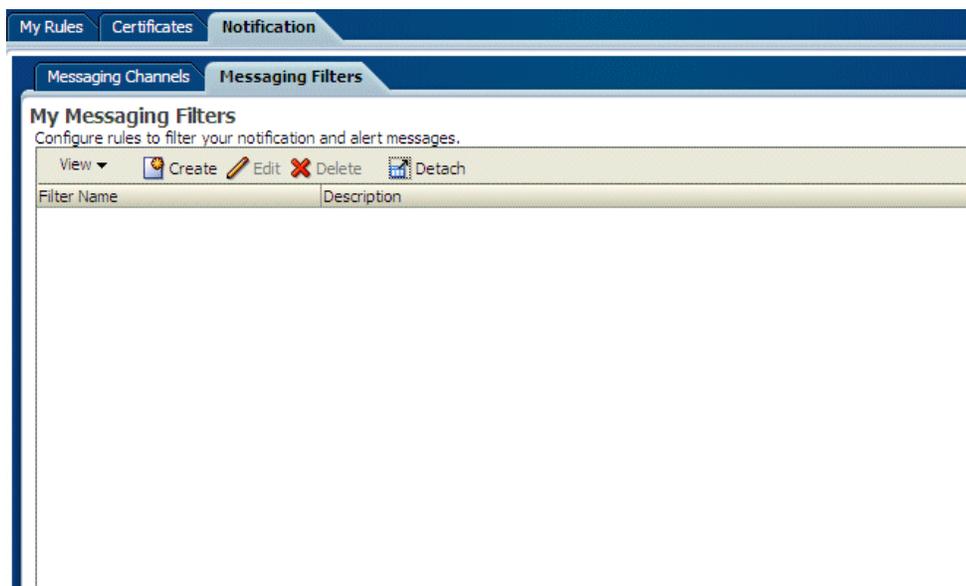
To view your messaging filters:

1. Click the **Notification** tab.
2. Click the **Messaging Filters** tab.

The My Messaging Filters list appears (Figure 27–43) and displays the following information:

- **Filter Name:** The name of the messaging filter
- **Description:** An optional description of the messaging filter

Figure 27–43 *Messaging Filters*



3. Click **View > Columns** and select the columns to display or hide.

You can also click **View > Reorder Columns** to display a dialog box to reorder the displayed columns.

27.9.4.2 Creating Messaging Filters

To create a messaging filter:

1. Click **Create**.

The Messaging Filters page appears, as shown in Figure 27–44.

Figure 27-44 Adding a Messaging Filter

The screenshot shows the 'Adding a Messaging Filter' dialog in Oracle BPM Worklist. The dialog is titled 'ORACLE BPM Worklist' and has a navigation bar with 'Home', 'Reports', 'Preferences', 'Help', and 'Logout'. The user is logged in as 'jstein'. The 'Notification' tab is active, and the 'Messaging Filters' sub-tab is selected. The dialog contains the following sections:

- Reference System Time:** Thursday, April 9, 2009 6:21:14 PM PDT (with OK and Cancel buttons).
- Filter Name:** A text input field.
- Description:** A text input field.
- Condition:**
 - Matching: All of the following conditions (dropdown).
 - Add Filter Condition: Status (dropdown) is Equal (dropdown) * (text input) + (Add button).
 - A table with columns: Attribute, Operator, Value, Value2 (if required), Delete.
- Action:**
 - Messaging Option: Send No Messages (dropdown).
 - Add Notification Channel: Instant Messaging (dropdown) + (Add button).
 - A table with columns: Channel, Address, Up, Down, Delete.

OK and Cancel buttons are located at the bottom right of the dialog.

2. Specify the following information:
 - **Filter Name:** The name of the messaging filter.
 - **Description:** An optional description for the messaging filter.
3. Define the filter conditions using the lists and fields in the **Condition** section, as follows:
 - a. Select whether notifications must meet all of the conditions or any of the conditions by selecting either the **All of the following conditions** or the **Any of the following conditions** options.
 - b. Select the attribute from the list.
 - c. Select the operator, such as **isEqual**, from the list.
 - d. Type the value of the condition in the text box.
 - e. Click **Add** to add the condition to the list.
 - f. Repeat these steps to add more filter conditions. To remove a filter condition, click **Delete**.
4. Select from the following messaging options in the **Action** section:
 - **Send No Messages:** Do not send a message to any channel.
 - **Send Messages to All Selected Channels:** Send a message to all specified channels in the address list.

- **Send to the First Available Channel:** Send a message serially to channels in the address list until one successful message is sent. This entails performing a send to the next channel when the current channel returns a failure status.
- 5. To set the delivery channel, select a channel from the **Add Notification Channel** list and click **Add**. To remove a channel, click **Delete**.
- 6. Use the up and down arrows to prioritize channels. If available, the top-most channel receives messages meeting the filter criteria if you select *Send to the First Available Channel*.
- 7. Click **OK**.

The messaging filter appears on the My Messaging Filters page. The My Messaging Filters page enables you to edit or delete the channel. Click **Cancel** to dismiss the dialog box without creating the filter.

27.9.4.3 Editing a Messaging Filter

To edit a messaging filter:

1. Select the channel on the My Messaging Filters page.
2. Click **Edit**.
3. Click **OK** to update the messaging filter. Click **Cancel** to dismiss the dialog box without modifying the filter.

27.9.4.4 Deleting a Messaging Filter

To delete a messaging filter:

1. Select the filter on the My Messaging Filters page.
2. Click **Delete**. A confirmation dialog appears.
3. Click **OK** to delete the messaging filter. Click **Cancel** to dismiss the dialog box without deleting the filter.

27.10 Using Flex Fields

Human workflow flex fields store and query use case-specific custom attributes. These custom attributes typically come from the task payload values. Storing custom attributes in flex fields provides the following benefits:

- They can be displayed as a column in the task listing
- They can filter tasks in custom views and advanced searches
- They can be used for a keyword-based search

For example the `Requestor`, `PurchaseOrderID`, and `Amount` fields in a purchase order request payload of a task can be stored in the flex fields. An approver logging into Oracle BPM Worklist can see these fields as column values in the task list and decide which task to access. The user can define views that filter tasks based on the flex fields. For example, a user can create views for purchase order approvals based on different amount ranges. If the user must also retrieve tasks at some point related to a specific requestor or a purchase order ID, they can specify this in the keyword field and perform a search to retrieve the relevant tasks.

For the flex fields to be populated, an administrator must create flex field mappings, as follows:

1. Specify a label for the flex field to be populated.

2. Map the payload attribute containing the data to the label.

These mappings are valid for a certain task type. Therefore, each task type can have different flex field mappings. After the mapping is complete and any new task is initiated, the value of the payload is promoted to the mapped flex field. Tasks initiated before the mapping do not contain the value in the flex field. Only top-level simple type attributes in the payload can be promoted to a flex field. Complex attributes or simple types nested inside a complex attribute cannot be promoted. It is important to define the payload for a task in the Human Task Editor, keeping in mind which attributes from the payload may need to be promoted to a flex field. All text and number flex fields are automatically included in the keyword-based search.

Essentially, the Human Task Editor is used only when defining the payload for a task. All other operations are performed at runtime.

Directory naming is not available concomitant with the flex file naming convention.

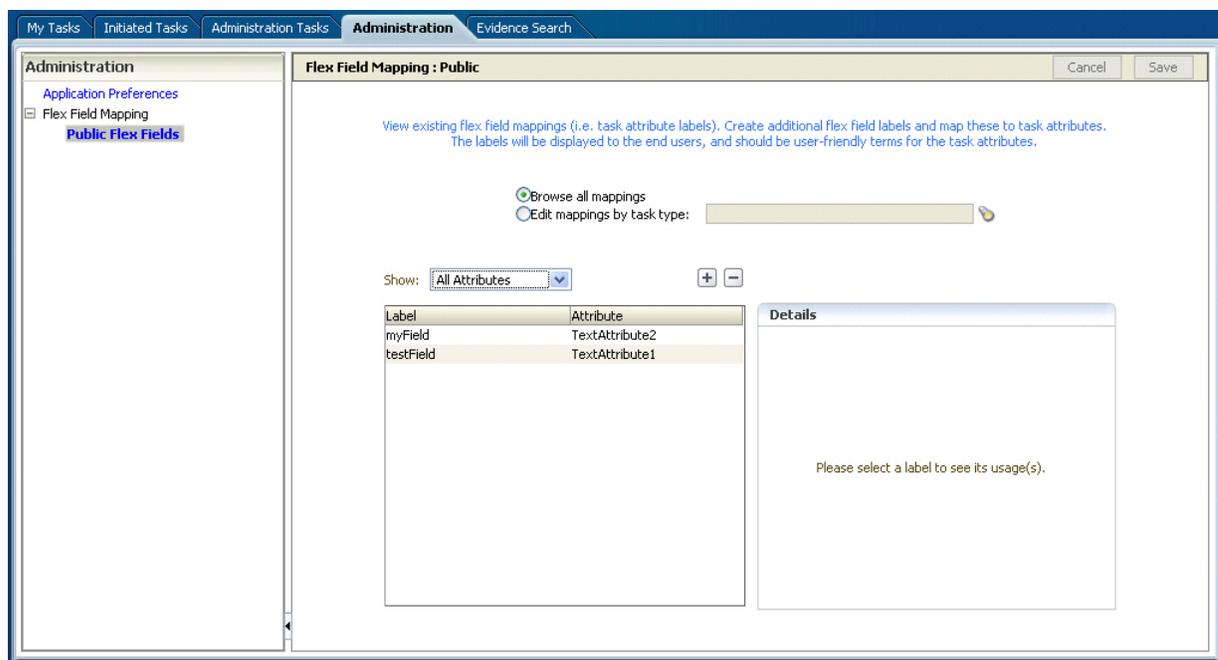
Note: Flex fields must be defined before instances of the business process are generated. Only instances generated after flex fields are created will reflect the correct flex fields. Older instances of the business process do not reflect subsequent flex field changes.

27.10.1 How To Map Flex Fields

An administrator, or users with special privileges, can use flex field mapping, shown in [Figure 27-45](#), to promote data from the payload to inline attribute flex fields. By promoting data to flex fields, the data becomes searchable and can be displayed as columns on the task list page.

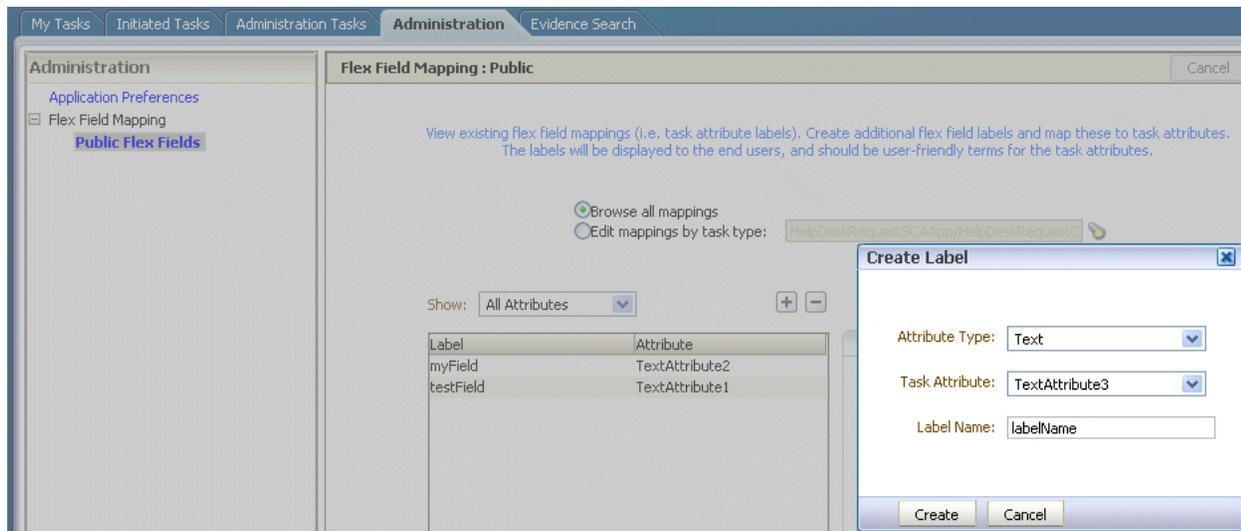
Administrators can map public flex fields. Users who have been granted the `workflow.mapping.publicFlexField` privilege can map public flex fields, and see a **Public Flex Fields** node on the **Administration** tab.

Figure 27-45 Flex Field Mapping



To create labels:

To create a flex field mapping, an administrator first defines a semantic label, which provides a more meaningful display name for the flex field attribute. Click **Add** to use the Create Label dialog, shown in [Figure 27–46](#).

Figure 27–46 *Creating a Label*

As the figure shows, **labelName** is mapped to the task attribute **TextAttribute3**. The payload attribute is also mapped to the label. In this example, the **Text** attribute type is associated with **labelName**. The result is that the value of the **Text** attribute is stored in the **TextAttribute3** column, and **labelName** is the column label displayed in the user's task list. Labels can be reused for different task types. You can delete a label only if it is not used in any mappings.

A mapped payload attribute can also be displayed as a column in a custom view, and used as a filter condition in both custom views and workflow rules. The display name of the payload attribute is the attribute label that is selected when doing the mapping.

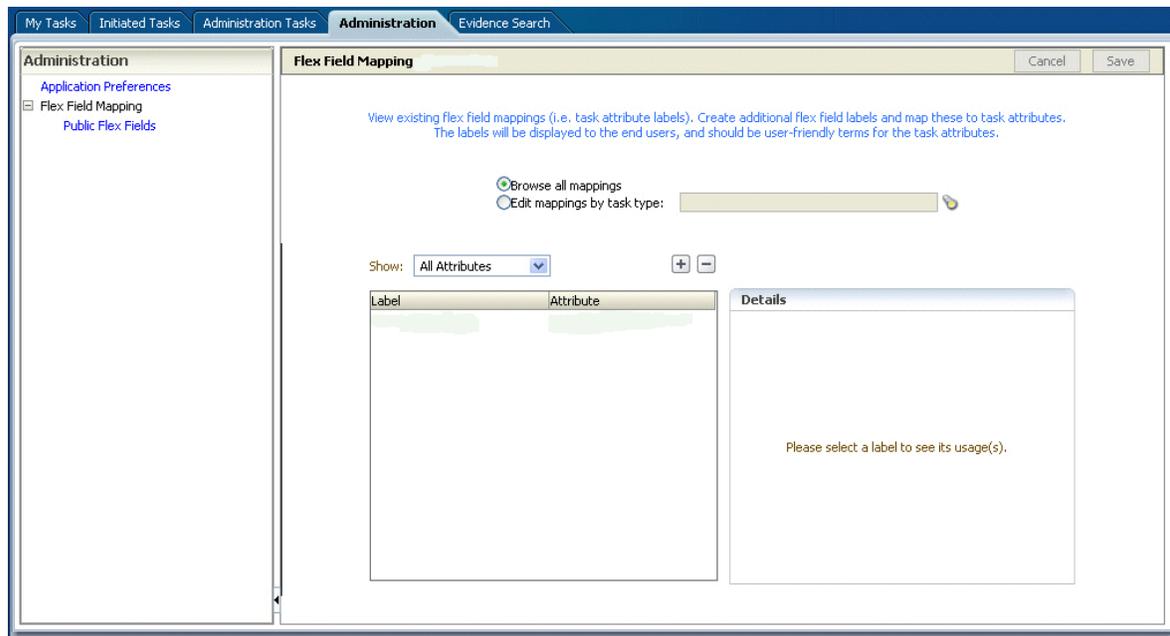
Note the following restrictions:

- Only simple type payload attributes can be mapped.
- A flex field (and thus a label) can be used only once per task type.
- Data type conversion is not supported for the number or date data types. For example, you may not map a payload attribute of type `string` to a label of type `number`.

To browse all mappings:

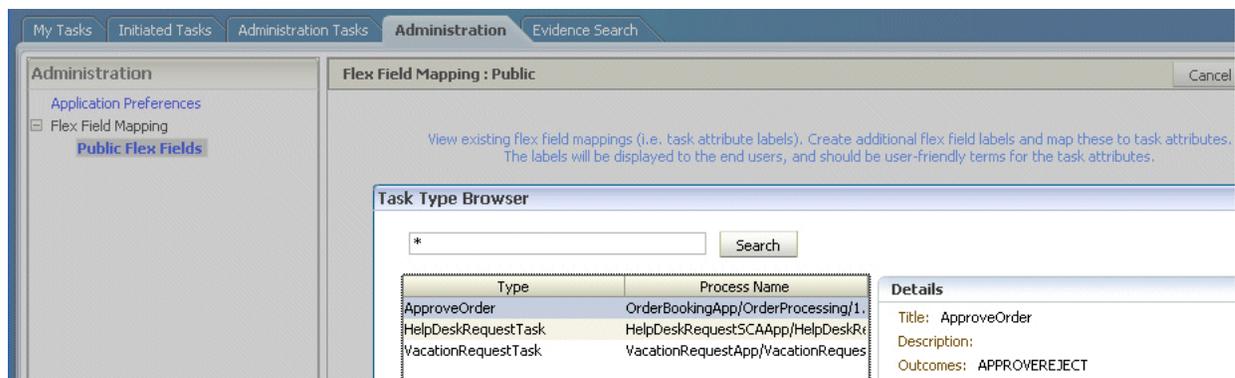
1. Click **Browse all mappings**.
2. Select a row in the label table to display all the payload attributes mapped to a particular label.

Figure 27–47 Browsing Mappings



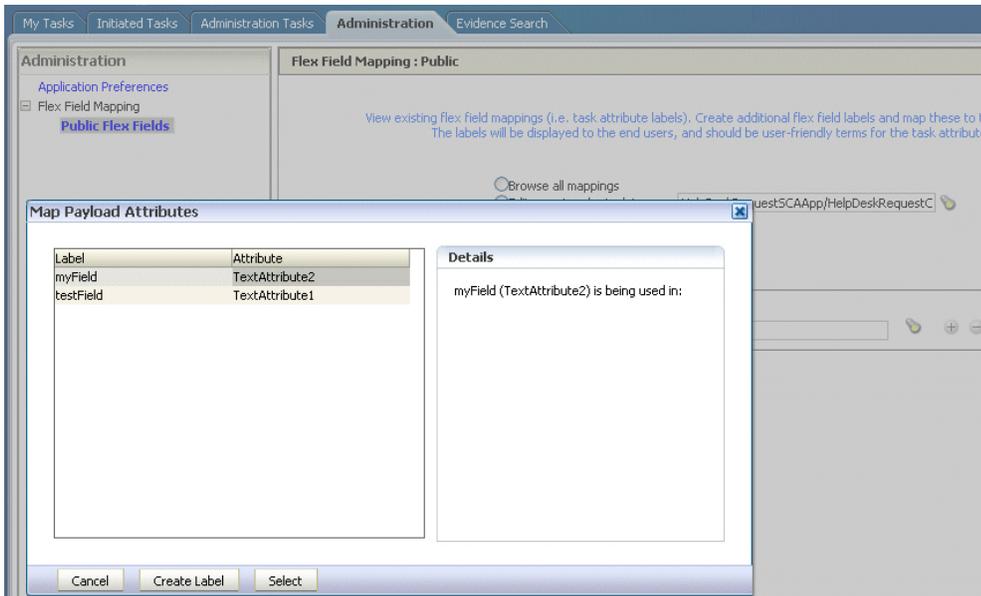
To edit mappings by task type:

1. Click **Edit mappings by task type**, optionally provide a task type, and click **Search**.
2. Select a task type and click **OK**.



3. With the task type displayed in the **Edit mappings by task type** field, click **Go**. All current mappings for the task type are displayed, as shown in [Figure 27–48](#).

Figure 27–48 *Selecting a Label*



4. Select a mapping label and click **Select**.

Figure 27–49 shows a completed mapping.

Figure 27–49 *Flex Field Mapping Created*



See [Section 29.1.9.1, "Internationalization of Attribute Labels"](#) for more information.

27.11 Creating Worklist Reports

[Table 27–10](#) lists the worklist reports available for task analysis.

Table 27–10 Worklist Report Types

Report Name	Description	Input Parameters
Unattended Tasks	Provides an analysis of tasks assigned to users' groups or reportees' groups that have not yet been acquired (the "unattended" tasks).	<ul style="list-style-type: none"> ■ Assignee—This option (required) selects tasks assigned to the user's group (My Group), tasks assigned to the reportee's groups (Reportees), tasks where the user is a creator (Creator), or tasks where the user is an owner (Owner). ■ Creation Date—An optional date range ■ Expiration Date—An optional date range ■ Task State—The state (optional) can be Any, Assigned, Expired, or Information Requested. ■ Priority—The priority (optional) can be Any, Highest, High, Normal, Low, or Lowest.
Tasks Priority	Provides an analysis of the number of tasks assigned to a user, reportees, or their groups, broken down by priority.	<ul style="list-style-type: none"> ■ Assignee—Depending on the assignee that you select, this required option includes tasks assigned to the logged-in user (My), tasks assigned to the user and groups that the user belongs to (My & Group), or tasks assigned to groups to which the user's reportees belong (Reportees). ■ Creation Date—An optional date range ■ Ended Date—An optional date range for the end dates of the tasks to be included in the report ■ Priority—The priority (optional) can be Any, Highest, High, Normal, Low, or Lowest.
Tasks Cycle Time	Provides an analysis of the time taken to complete tasks from assignment to completion based on users' groups or reportees' groups.	<ul style="list-style-type: none"> ■ Assignee—Depending on the assignee that you select, this required option includes your tasks (My) or tasks assigned to groups to which your reportees belong (Reportees). ■ Creation Date—An optional date range ■ Ended Date—An optional date range for the end dates of the tasks to be included in the report ■ Priority—The priority (optional) can be Any, Highest, High, Normal, Low, or Lowest.
Tasks Productivity	Provides an analysis of assigned tasks and completed tasks in a given time period for a user, reportees, or their groups.	<ul style="list-style-type: none"> ■ Assignee—Depending on the assignee that the user selects, this required option includes the user's tasks (My & Group) or tasks assigned to groups to which the user's reportees belong (Reportees). ■ Creation Date (range)—An optional creation date range. The default is one week. ■ Task Type—Use the Search (flashlight) icon to select from a list of task titles. All versions of a task are listed on the Select Workflow Task Type page (optional).
Tasks Time Distribution	Provides the time an assignee takes to perform a task.	<ul style="list-style-type: none"> ■ Assignee—Depending on the assignee that the user selects, this required option includes the user's tasks (My & Group) or tasks assigned to groups to which the user's reportees belong (Reportees). ■ From...to (date range)—An optional creation date range. The default is one week. ■ Task Type—Use the Search (flashlight) icon to select from a list of task titles. All versions of a task are listed on the Select Workflow Task Type page (optional).

27.11.1 How To Create Reports

Reports are available from the **Reports** link. Report results cannot be saved.

To create a report:

1. Click the **Reports** link.
2. Click the type of report you want to create.

[Figure 27–50](#) shows the report types available.

Figure 27–50 Oracle BPM Worklist Reports



3. Provide inputs to define the search parameters of the report.

Figure 27–51 shows an example of the Unattended Tasks Report input page. The other reports are similar. See Table 27–10 for information about input parameters for all the report types.

Figure 27–51 Unattended Tasks Report—Input Page for Task Analysis

Unattended Tasks Report
 Provides an analysis of tasks assigned to users' groups or reportees' groups that need attention because they have not yet been acquired

Assignee:

Creation Date: to

Expiration Date: to

Task State:

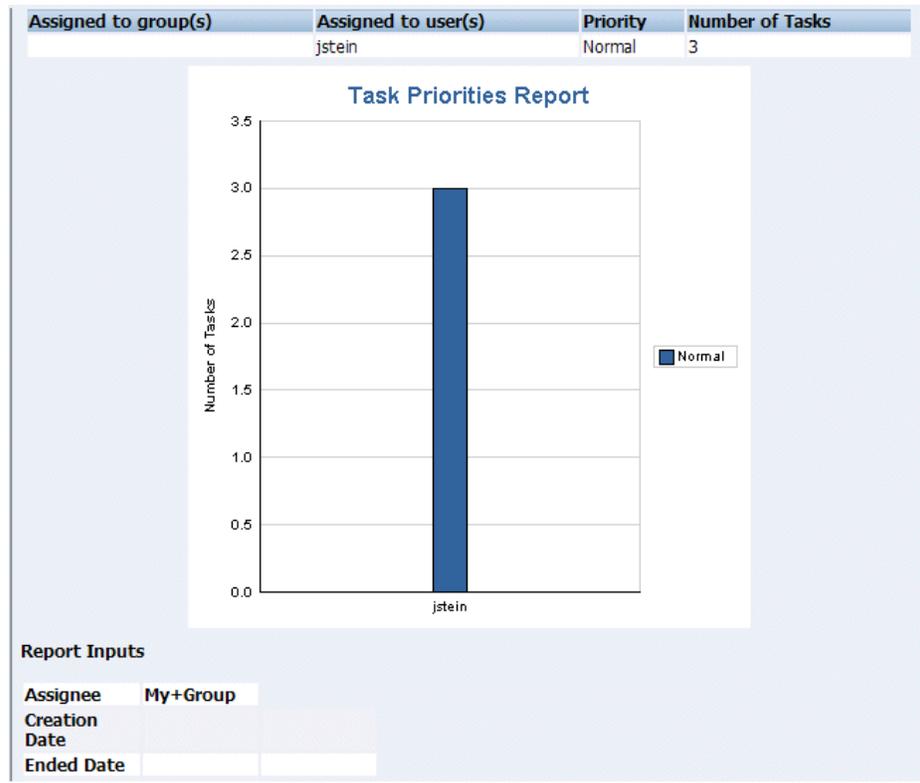
Priority:

4. Click Run.

27.11.2 What Happens When You Create Reports

As shown in Figure 27–52, report results (for all report types) are displayed in both a table format and a bar chart format. The input parameters used to run the report are displayed under **Report Inputs**, in the lower-left corner (may require scrolling to view).

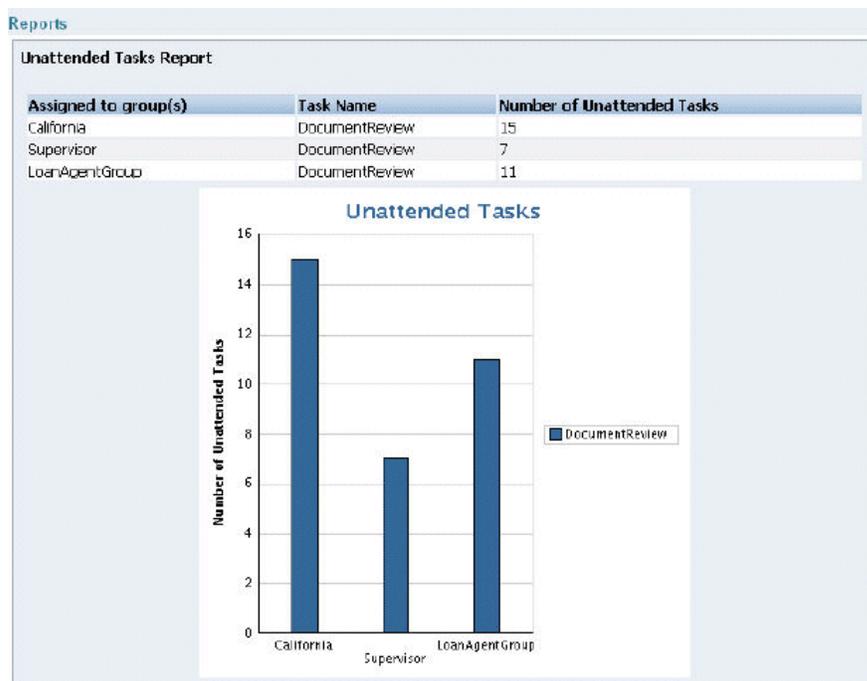
Figure 27–52 Report Display—Table Format, Bar Chart Format, and Report Inputs



27.11.2.1 Unattended Tasks Report

Figure 27–53 shows an example of an Unattended Tasks report.

Figure 27–53 Unattended Tasks Report

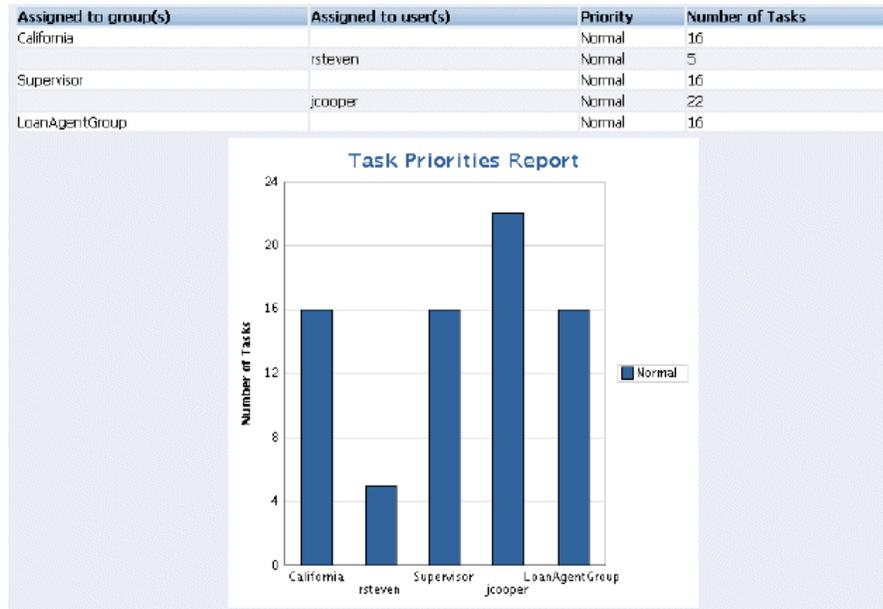


The report shows that the California group has 15 unattended tasks, the Supervisor group has 7 unattended tasks, and the LoanAgentGroup has 11 unattended tasks. The unattended (unclaimed) tasks in this report are all DocumentReview tasks. If multiple types of unattended task exists when a report is run, all task types are included in the report, and the various task types are differentiated by color.

27.11.2.2 Tasks Priority Report

Figure 27–54 shows an example of a Tasks Priority report.

Figure 27–54 Tasks Priority Report

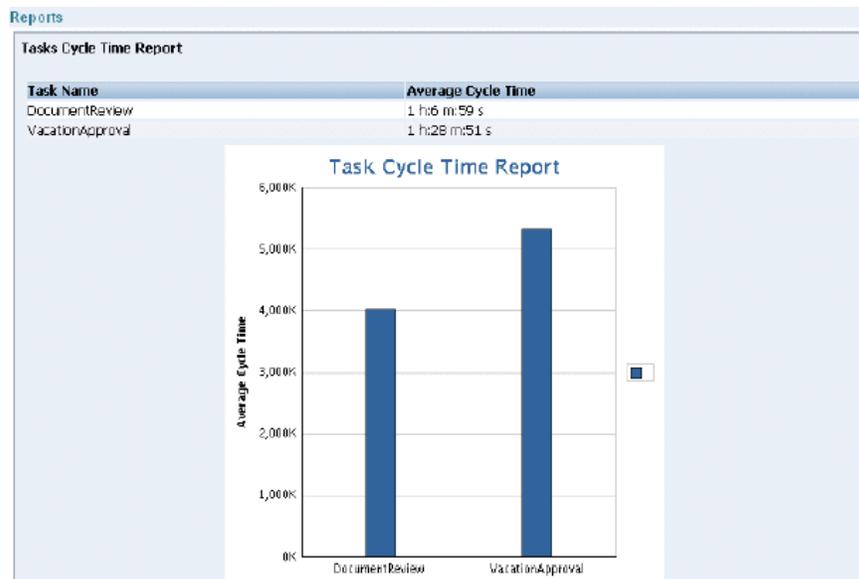


The report shows that the California group, the Supervisor group, and the LoanAgentGroup each have 16 tasks of normal priority. The users rsteven and jcooper have 5 and 22 tasks, respectively, all normal priority. Priorities (highest, high, normal, low, lowest) are distinguished by different colors in the bar chart.

27.11.2.3 Tasks Cycle Time Report

Figure 27–55 shows an example of a Tasks Cycle Time Report.

Figure 27–55 Tasks Cycle Time Report

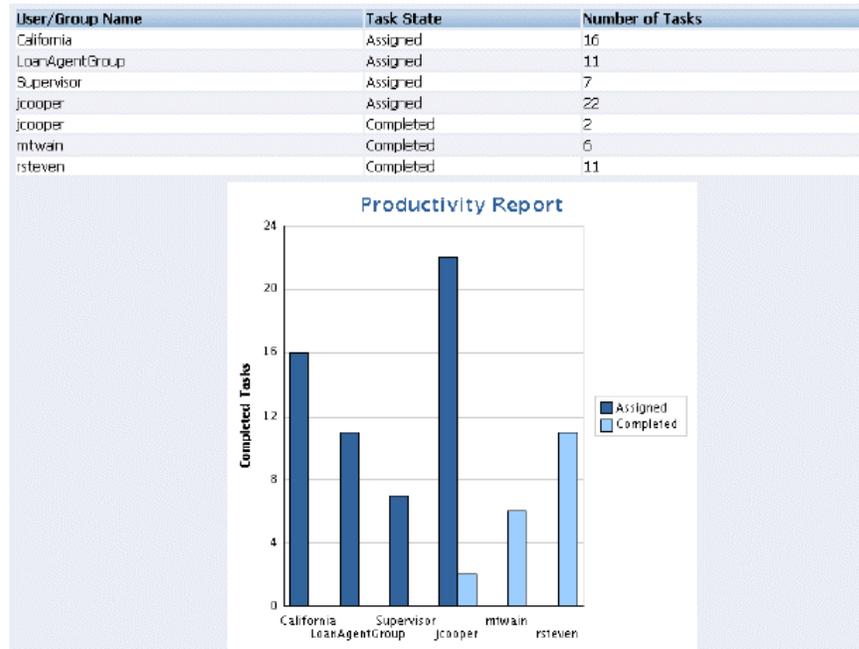


The report shows that it takes 1 hour and 6 minutes on average to complete DocumentReview tasks, and 1 hour and 28 minutes on average to complete VacationApproval tasks. The bar chart shows the average cycle time in milliseconds.

27.11.2.4 Tasks Productivity Report

Figure 27–56 shows an example of a Tasks Productivity Report.

Figure 27–56 Tasks Productivity Report



The report shows the number of tasks assigned to the California, LoanAgentGroup, and Supervisor groups. For individual users, the report shows that jcooper has 22 assigned tasks. In addition to his assigned tasks, jcooper has completed 2 tasks. The

report shows that mtwain and rsteven have completed 6 and 11 tasks respectively. In the bar chart, the two task states—assigned and completed—are differentiated by color.

27.12 Accessing Oracle BPM Worklist in Local Languages

The identity service determines a user's preferred language and time zone. This information is extracted from either the JAZN file-based community or from an external directory service such as Oracle Internet Directory. If no preference information is available, then the user's preferred language and time zone are set to the system defaults, `en-US` and `America/Los_Angeles`. If an LDAP-based provider such as OID is used, then language settings are changed in the OID community.

When a user logs in, the worklist pages are rendered in the browser's locale and time zone. Most strings in the worklist come from the Worklist Application bundle. By default, this is the class

```
oracle.bpel.services.workflow.resource.WorkflowResourceBundle
```

However, this can be changed to a custom resource bundle by setting the appropriate application preference (see [Section 27.8.2, "How To Set the Worklist Display \(Application Preferences\)"](#)) or by providing an updated version of the default bundle class. See the Workflow Customizations sample for details.

For task attribute names, flex field attribute labels, and dynamic assignment function names, the strings come from configuring the resource property file `WorkflowLabels.properties`. This file exists in the `wfresource` subdirectory of the services config directory. See [Chapter 29, "Introduction to Human Workflow Services"](#) for information on adding entries to this file for dynamic assignment functions and attribute labels.

For custom actions and task titles, the display names come from the message bundle specified in the task configuration file. If no message bundle is specified, then the values specified at design time are used. See [Chapter 29, "Introduction to Human Workflow Services"](#) for information on how to specify message bundles so that custom actions and task titles are displayed in the preferred language.

27.12.1 How To Change the Language Used in the Worklist

The following is based on extracting a user's preferred language from a JAZN XML file.

To change the language:

Change the portion in bold to set the user's preferred language.

```
<preferredLanguage>en-US</preferredLanguage>
```

Oracle BPM Worklist supports the languages shown in [Table 27–11](#).

Table 27–11 Languages Supported in Oracle BPM Worklist

Language	Format
English	(en)
English (United States)	(en-US)
German	(de)
Spanish (International)	(es)

Table 27–11 (Cont.) Languages Supported in Oracle BPM Worklist

Language	Format
French	(fr)
Italian	(it)
Japanese	(ja)
Korean	(ko)
Portuguese (Brazil)	(pt-BR)
Chinese (Simplified)	(zh-CN)
Chinese (Traditional)	(zh-TW)

27.12.2 How To Change the Time Zone Used in the Worklist

The following is based on extracting a user's time zone from a JAZN XML file.

To change the time zone:

Change the string in bold to set the user's preferred time zone.

```
<timeZone>America/Los_Angeles</timeZone>
```

The format of the time zone string is *Continent/Region*. You can find the time zone values in the `$JAVA_HOME/jre/lib/zi` directory. The directories specify the continent names, for example Africa, Asia, America, and so on, while the files within the directories specify the regions. Note that some regions include sub-regions, for example `America/Indiana/Indianapolis`.

Building a Custom Worklist Client

Starting with the sample Worklist Application, you can build clients for workflow services using the APIs exposed by the workflow service. The APIs enable clients to communicate with the workflow service using local and remote EJBs, SOAP, and HTTP.

This chapter contains the following topics:

- [Section 28.1, "Introduction to Building Clients for Workflow Services"](#)
- [Section 28.2, "Packages and Classes for Building Clients"](#)
- [Section 28.3, "Workflow Service Clients"](#)
- [Section 28.4, "Class Paths for Clients Using SOAP"](#)
- [Section 28.5, "Class Paths for Clients Using Remote EJBs"](#)
- [Section 28.6, "Class Paths for Clients Using Local EJBs"](#)
- [Section 28.7, "Enterprise JavaBeans References in Web Applications"](#)
- [Section 28.8, "Initiating a Task"](#)
- [Section 28.9, "Changing Workflow Standard View Definitions"](#)
- [Section 28.10, "Writing a Worklist Application Using the HelpDeskUI Sample"](#)

28.1 Introduction to Building Clients for Workflow Services

The typical sequence of calls when building a simple worklist application is as follows.

To build a simple worklist application:

1. Get a handle to `IWorklistServiceClient` from `WorkflowServiceClientFactory`.
2. Get a handle to `ITaskQueryService` from `IWorklistServiceClient`.
3. Authenticate a user by passing a username and password to the `authenticate` method on `ITaskQueryService`. Get a handle to `IWorkflowContext`.
4. Query the list of tasks using `ITaskQueryService`.
5. Get a handle to `ITaskService` from `IWorklistServiceClient`.
6. Iterate over the list of tasks returned, performing actions on the tasks using `ITaskService`.

Example 28–1 demonstrates how to build a client for workflow services. A list of all tasks assigned to `jstein` is queried. A task whose outcome has not been set is approved.

Example 28–1 Building a Client for Workflow Services—Setting the Outcome to Approved

```

try
{
//Create JAVA WorkflowServiceClient
IWorkflowServiceClient wfSvcClient = WorkflowServiceClientFactory.getWorkflowServiceClient(
    WorkflowServiceClientFactory.REMOTE_CLIENT);
//Get the task query service
ITaskQueryService querySvc = wfSvcClient.getTaskQueryService();

//Login as jstein
IWorkflowContext ctx = querySvc.authenticate("jstein","welcome1".toCharArray(),null);
//Set up list of columns to query
List queryColumns = new ArrayList();
queryColumns.add("TASKID");
queryColumns.add("TASKNUMBER");
queryColumns.add("TITLE");
queryColumns.add("OUTCOME");

//Query a list of tasks assigned to jstein
List tasks = querySvc.queryTasks(ctx,
    queryColumns,
    null, //Do not query additional info
    ITaskQueryService.AssignmentFilter.MY,
    null, //No keywords
    null, //No custom predicate
    null, //No special ordering
    0, //Do not page the query result
    0);
//Get the task service
ITaskService taskSvc = wfSvcClient.getTaskService();
//Loop over the tasks, outputting task information, and approving any
//tasks whose outcome has not been set...
for(int i = 0 ; i < tasks.size() ; i ++)
{
    Task task = (Task)tasks.get(i);
    int taskNumber = task.getSystemAttributes().getTaskNumber();
    String title = task.getTitle();
    String taskId = task.getSystemAttributes().getTaskId();
    String outcome = task.getSystemAttributes().getOutcome();
    if(outcome == null)
    {
        outcome = "APPROVED";
        taskSvc.updateTaskOutcome(ctx,taskId,outcome);
    }
    System.out.println("Task #"+taskNumber+" ("+"title+") is "+outcome);
}
}
catch (Exception e)
{
//Handle any exceptions raised here...
System.out.println("Caught workflow exception: "+e.getMessage());
}
}

```

28.2 Packages and Classes for Building Clients

Use the following packages and classes for building clients:

- oracle.bpel.services.workflow.metadata.config.model

The classes in this package contain the object model for the workflow configuration in the task definition file. The `ObjectFactory` class can be used to create objects.

- `oracle.bpel.services.workflow.metadata.routingslip.model`
The classes in this package contain the object model for the routing slip. The `ObjectFactory` class can be used to create objects.
- `oracle.bpel.services.workflow.metadata.taskdisplay.model`
The classes in this package contain the object model for the task display. The `ObjectFactory` class can be used to create objects.
- `oracle.bpel.services.workflow.metadata.taskdefinition.model`
The classes in this package contain the object model for the task definition file. The `ObjectFactory` class can be used to create objects.
- `oracle.bpel.services.workflow.client.IWorkflowServiceClient`
The interface for the workflow service client.
- `oracle.bpel.services.workflow.client.WorkflowServiceClientFactory`
The factory for creating the workflow service client.
- `oracle.bpel.services.workflow.metadata.ITaskMetadataService`
The interface for the task metadata service.
- `oracle.bpel.services.workflow.task.ITaskService`
The interface for the task service.
- `oracle.bpel.services.workflow.task.IRoutingSlipCallback`
The interface for the callback class to receive callbacks during task processing.
- `oracle.bpel.services.workflow.task.IAssignmentService`
The interface for the assignment service.

28.3 Workflow Service Clients

Any worklist application accesses the various workflow services through the workflow service client. The workflow service client code encapsulates all the logic required for communicating with the workflow services using different local and remote protocols. After the worklist application has an instance of the workflow service client, it does not need to consider how the client communicates with the workflow services.

The advantages of using the client are as follows:

- Hides the complexity of the underlying connection mechanisms such as SOAP/HTTP and Enterprise JavaBeans
- Facilitates changing from using one particular invocation mechanism to another, for example from SOAP/HTTP to remote Enterprise JavaBeans

The following class is used to create instances of the `IWorkflowServiceClient` interface:

```
oracle.bpel.services.workflow.client.WorkflowServiceClientFactory
```

WorkflowServiceClientFactory has several methods that create workflow clients. The simplest method, `getWorkflowServiceClient`, takes a single parameter, the client type. The client type can be one of the following:

- `WorkflowServiceClientFactory.LOCAL_CLIENT`—The client uses a local Enterprise JavaBeans interface to invoke the workflow services.
- `WorkflowServiceClientFactory.REMOTE_CLIENT`—The client uses a remote Enterprise JavaBeans interface to invoke workflow services located remotely from the client.
- `WorkflowServiceClientFactory.SOAP_CLIENT`—The client uses SOAP to invoke web service interfaces to the workflow services, located remotely from the client.

The other factory methods enable you to specify the connection properties directly (rather than having the factory load them from the `wf_client_config.xml` file), and enable you to specify a logger to log client activity.

The following enhancements to the workflow service clients are included in this release:

- You can specify the workflow client configuration using either a JAXB object or a map, as shown in [Example 28-2](#) and [Example 28-3](#).

Example 28-2 Workflow Client Configuration Using a JAXB Object

```
WorkflowServicesClientConfigurationType wsct = new WorkflowServicesClientConfigurationType();
List<ServerType> servers = wsct.getServer();
ServerType server = new ServerType();
server.setDefault(true);
server.setName(serverName);
servers.add(server);

RemoteClientType rct = new RemoteClientType();
rct.setServerURL("t3://stapj73:7001");
rct.setUserName("weblogic");
rct.setPassword("weblogic");
rct.setInitialContextFactory("weblogic.jndi.WLInitialContextFactory");
rct.setParticipateInClientTransaction(false);
server.setRemoteClient(rct);
IWorkflowServiceClient wfSvcClient = WorkflowServiceClientFactory.getWorkflowServiceClient(
    WorkflowServiceClientFactory.REMOTE_CLIENT, wsct, logger);
```

Example 28-3 Workflow Client Configuration Using a Map

```
Map<IWorkflowServiceClientConstants.CONNECTION_PROPERTY, java.lang.String> properties = new
    HashMap<IWorkflowServiceClientConstants.CONNECTION_PROPERTY, java.lang.String>();

properties.put(IWorkflowServiceClientConstants.CONNECTION_PROPERTY.MODE,
    IWorkflowServiceClientConstants.MODE_DYNAMIC);

properties.put(IWorkflowServiceClientConstants.CONNECTION_PROPERTY.SOAP_END_POINT_ROOT,
    "http://localhost:8888");

IWorkflowServiceClient client =
    WorkflowServiceClientFactory.getWorkflowServiceClient(WorkflowServiceClientFactory.SOAP_CLIENT,
    properties, null);
```

- Clients can optionally pass in a `java.util.logging.Logger` where the client logs messages. If no logger is specified, then the workflow service client code does

not log anything. [Example 28–4](#) shows how a logger can be passed to the workflow service clients.

Example 28–4 Passing a Logger to the Workflow Service Clients

```
java.util.logging.Logger logger = ....;

IWorkflowServiceClient client =
    WorkflowServiceClientFactory.getWorkflowServiceClient(WorkflowServiceClientFactory.REMOTE_CLIENT,
        properties, logger);
```

Through the factory, it is possible to get the client libraries for all the workflow services. See [Table 29–1, "Enterprise JavaBeans, SOAP, and Java Support"](#) for the clients available for each of the services.

Note that you can obtain instances of `BPMIdentityService` and `BPMIdentityConfigService` by calling the `getSOAPIdentityServiceClient` and `getSOAPIdentityConfigServiceClient` methods on `WorkflowServiceClientFactory`. You can obtain all other services through an instance of `IWorkflowServiceClient`.

The client classes use the configuration file `wf_client_config.xml` for the service end points. In the client class path, this file is in the class path directly, meaning the containing directory is in the class path. The `wf_client_config.xml` file contains:

- A section for remote clients:

```
<remoteClient>
    <serverURL>t3://host_name.domain_name:7001</serverURL>
    <userName>weblogic</userName>
    <password>weblogic</password>
    <initialContextFactory>weblogic.jndi.WLInitialContextFactory
    </initialContextFactory>
    <participateInClientTransaction>>false</participateInClientTransaction>
</remoteClient>
```

- A section for SOAP end points for each of the services:

```
<soapClient>
    <rootEndPointURL>http://host_name.domain_name:7001</rootEndPointURL>
    <identityPropagation mode="dynamic" type="saml">
    <policy-references>
        <policy-reference enabled="true" category="security"
            uri="oracle/wss10_saml_token_client_policy"/>
    </policy-references>
    </identityPropagation>
</soapClient>
```

The workflow client configuration XML schema definition is in the `wf_client_config.xsd` file.

28.3.1 The `IWorkflowServiceClient` Interface

The `IWorkflowServiceClient` interface provides methods, summarized in [Table 28–1](#), for obtaining handles to the various workflow services interfaces.

Table 28–1 *IWorkflowServiceClient Methods*

Method	Interface
getTaskService	oracle.bpel.services.workflow.task.ITaskService
getTaskQueryService	oracle.bpel.services.workflow.query.ITaskQueryService
getTaskReportService	oracle.bpel.services.workflow.report.ITaskReportService
getTaskMetadataService	oracle.bpel.services.workflow.metadata.ITaskMetadataService
getUserMetadataService	oracle.bpel.services.workflow.user.IUserMetadataService
getRuntimeConfigService	oracle.bpel.services.workflow.runtimeconfig.IRuntimeConfigService
getTaskEvidenceService	oracle.bpel.services.workflow.metadata.ITaskMetadataService

28.4 Class Paths for Clients Using SOAP

SOAP clients must have the following JAR files in their class path:

- `${bea.home}/wlserver_10.3/server/lib/wlfullclient.jar`
- `${bea.home}/AS11gR1SOA/webservices/wsclient_extended.jar`
- `${bea.home}/AS11gR1SOA/soa/modules/oracle.soa.fabric_11.1.1/bpm-infra.jar`
- `${bea.home}/AS11gR1SOA/soa/modules/oracle.soa.workflow_11.1.1/bpm-services.jar`

You can generate the `wlfullclient.jar` file using the following commands:

```
cd ${bea.home}/wlserver_10.3/server/lib
java -jar ../../../../modules/com.bea.core.jarbuilder_1.3.0.0.jar
```

Note: Client applications no longer use the `system\services\config` or `system\services\schema` directories in the class path.

28.5 Class Paths for Clients Using Remote EJBs

Clients using remote EJBs must have the following JAR files in their class path:

- `xmlparserv2.jar`
- `xml.jar`
- `bpm-infra.jar`
- `bpm-services.jar`
- `bpm-services-client.jar` (only if you are using the ADF data controls for workflow)

Note: Client applications no longer use the `system\services\config` or `system\services\schema` directories in the class path.

28.6 Class Paths for Clients Using Local EJBs

Only applications running as part of the soa-infra application or those that are a child application of the soa-infra application can use local EJBs. In either case, the child application has all the necessary classes in its class path, either because they are part of soa-infra or because they inherit the class path as the child of soa-infra.

Note: Client applications no longer use the `system\services\config` or `system\services\schema` directories in the class path.

28.7 Enterprise JavaBeans References in Web Applications

If a web application uses the workflow service local EJBs, then the client application must do the following:

- The application must be a child application of the `hw_services` application.
- The application must define the Enterprise JavaBeans local references in its `web.xml` file. The local references for each of the services are shown in [Example 28–5](#) and [Example 28–6](#).

Example 28–5 Task Service

```
<ejb-local-ref id="EjbRef_TaskServiceBean_Message">
  <ejb-ref-name>ejb/local/TaskServiceBean</ejb-ref-name>
  <ejb-ref-type>Session</ejb-ref-type>
  <local-home>oracle.bpel.services.workflow.task.ejb.TaskServiceLocalHome</local-home>
  <local>oracle.bpel.services.workflow.task.ejb.TaskServiceLocal</local>
  <ejb-link>TaskServiceBean</ejb-link>
</ejb-local-ref>
```

Example 28–6 Task Metadata Service

```
<ejb-local-ref id="EjbRef_TaskMetadataServiceBean_Message">
  <ejb-ref-name>ejb/local/TaskMetadataServiceBean</ejb-ref-name>
  <ejb-ref-type>Session</ejb-ref-type>
  <local-home>oracle.bpel.services.workflow.metadata.ejb.TaskMetadataServiceLocalHome</local-home>
  <local>oracle.bpel.services.workflow.metadata.ejb.TaskMetadataServiceLocal</local>
  <ejb-link>TaskMetadataServiceBean</ejb-link>
</ejb-local-ref>
```

Note: Only child applications can use local EJBs. This restricts standalone Java clients to using either remote EJBs or SOAP clients.

See [Chapter 29, "Introduction to Human Workflow Services,"](#) for more information on `TaskQueryService`, `TaskReportService`, `UserMetadataService`, and `RuntimeConfigService`.

28.8 Initiating a Task

Tasks can be initiated programmatically, in which case the following task attributes must be set:

- `taskDefinitionId`
- `title`

- payload
- priority

The following task attributes are optional, but are typically set by clients:

- creator
- ownerUser—Defaults to bpeladmin if empty
- processInfo
- identificationKey—Tasks can be queried based on the identification key from the TaskQueryService.

28.8.1 Creating a Task

The task object model is available in the package

```
oracle.bpel.services.workflow.task.model
```

To create objects in this model, use the `ObjectFactory` class.

28.8.2 Creating a Payload Element in a Task

The task payload can contain multiple payload message attributes. Since the payload is not well defined until the task definition, the Java object model for the task does not contain strong type objects for the client payload. The task payload is represented by the `AnyType` Java object. The `AnyType` Java object is created with an XML element whose root is `payload` in the namespace

```
http://xmlns.oracle.com/bpel/workflow/task
```

The payload XML element contains all the other XML elements in it. Each XML element defines a message attribute.

[Example 28–7](#) shows how to set a task payload.

Example 28–7 Setting a Task Payload

```
import oracle.bpel.services.workflow.task.model.AnyType;
import oracle.bpel.services.workflow.task.model.ObjectFactory;
import oracle.bpel.services.workflow.task.model.Task;
.....

Document document = //createXMLDocument
Element payloadElem = document.createElementNS("http://xmlns.oracle.com/bpel/workflow/
    task", "payload");
Element orderElem = document.createElementNS("http://xmlns.oracle.com/pcbpel/test/order", "order");
Element child = document.createElementNS("http://xmlns.oracle.com/pcbpel/test/order", "id");
    child.appendChild(document.createTextNode("1234567"));
    orderElem.appendChild(child);
    payloadElem.appendChild(orderElem);
    document.appendChild(payloadElem);

task.setPayloadAsElement(payloadElem);
```

Note: The `AnyType.getContent()` element returns an unmodifiable list of XML elements. You cannot add other message attributes to the list.

28.8.3 Initiating a Task Programmatically

[Example 28–8](#) shows how to initiate a vacation request task programmatically.

Example 28–8 *Initiating a Vacation Request Task Programmatically*

```
// create task object
ObjectFactory objectFactory = new ObjectFactory();
Task task = objectFactory.createTask();

// set title
task.setTitle("Vacation request for jcooper");

// set creator
task.setCreator("jcooper");

// set taskDefinitionId
task.setTaskDefinitionId("/VacationRequestApp/VacationRequest!1.0*2007-04-26-10-49-50/
  VacationRequest"); (Your task definition ID will be different.)

// create and set payload
Document document = XMLUtil.createDocument();
Element payloadElem = document.createElementNS(TASK_NS, "payload");
Element vacationRequestElem = document.createElementNS(VACATION_REQUEST_NS,
  "VacationRequestProcessRequest");

Element creatorChild = document.createElementNS(VACATION_REQUEST_NS, "creator");
creatorChild.appendChild(document.createTextNode("jcooper"));
vacationRequestElem.appendChild(creatorChild);

Element fromDateChild = document.createElementNS(VACATION_REQUEST_NS, "fromDate");
fromDateChild.appendChild(document.createTextNode("2006-08-05T12:00:00"));
vacationRequestElem.appendChild(fromDateChild);

Element toDateChild = document.createElementNS(VACATION_REQUEST_NS, "toDate");
toDateChild.appendChild(document.createTextNode("2006-08-08T12:00:00"));
vacationRequestElem.appendChild(toDateChild);

Element reasonChild = document.createElementNS(VACATION_REQUEST_NS, "reason");
reasonChild.appendChild(document.createTextNode("Hunting"));
vacationRequestElem.appendChild(reasonChild);

payloadElem.appendChild(vacationRequestElem);
document.appendChild(payloadElem);

task.setPayloadAsElement(payloadElem);

IWorkflowServiceClient workflowServiceClient =
  WorkflowServiceClientFactory.getWorkflowServiceClient
    (WorkflowServiceClientFactory.SOAP_CLIENT);
ITaskService taskService = workflowServiceClient.getTaskService();
IInitiateTaskResponse iInitiateTaskResponse = taskService.initiateTask(task);
Task retTask = iInitiateTaskResponse.getTask();
System.out.println("Initiated: " + retTask.getSystemAttributes().getTaskNumber() + " - " +
  retTask.getSystemAttributes().getTaskId());
return retTask;
```

28.9 Changing Workflow Standard View Definitions

The worklist application and the `UserMetadataService` API provide methods that you can use to create, update, and delete standard views. See [Section 29.1.7, "User Metadata Service"](#) for more information.

28.10 Writing a Worklist Application Using the HelpDeskUI Sample

The following example shows how to modify the help desk interface that is part of the `HelpDeskRequest` demo.

To write a worklist application

1. Create the workflow context by authenticating the user.

```
// get workflow service client
IWorkflowServiceClient wfSvcClient =
    WorkflowServiceClientFactory.getWorkflowServiceClient
    (WorkflowServiceClientFactory.REMOTE_CLIENT);

//get the workflow context
IWorkflowContext wfCtx =
wfSvcClient.getTaskQueryService().authenticate(userId, pwd, null);
```

This is Step 3 in [Section 28.1, "Introduction to Building Clients for Workflow Services."](#)

The `login.jsp` file of `HelpDeskRequest` uses the preceding API to authenticate the user and create a workflow context. After the user is authenticated, the `statusPage.jsp` file displays the tasks assigned to the logged-in user. [Example 28-9](#) shows sample code from the `login.jsp` file.

Example 28-9 Login.jsp

```
<%@ page import="javax.servlet.http.HttpSession"
import="oracle.bpel.services.workflow.client.IWorkflowServiceClient"
import="oracle.bpel.services.workflow.client.WorkflowServiceClientFactory"
import="java.util.Set"
import="java.util.Iterator"
import="oracle.bpel.services.workflow.verification.IWorkflowContext"
import="oracle.tip.pc.services.identity.config.ISConfiguration"%>
<%@ page contentType="text/html; charset=windows-1252"%>

<html>
<head>
<title>Help desk request login page</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
</head>

<body bgcolor="#F0F0F0" text="#000000" style="font: 12px verdana; line-height:18px">
<center>
<div style="width:640px;padding:15px;border-width: 10px; border-color: #87b4d9; border-style:
solid;
background-color:white; text-align:left">

    <!-- Page Header, Application banner, logo + user status -->
    <jsp:include page="banner.jsp"/>

    <!-- Initiate Meta Information -->
```

```

<div style="background-color:#F0F0F0; border-top:10px solid white;border-bottom:
  10px solid white;padding:10px;text-align:center" >
<b>Welcome to the HelpDesk application</b>
</div>

<%
String redirectPrefix = "/HelpDeskUI/";
// Ask the browser not to cache the page
response.setHeader("Pragma", "no-cache");
response.setHeader("Cache-Control", "no-cache");

HttpSession httpSession = request.getSession(false);
if (httpSession != null) {

    IWorkflowContext ctx = (IWorkflowContext) httpSession.getAttribute("workflowContext");
    if (ctx != null) {
        response.sendRedirect(redirectPrefix + "statusPage.jsp");
    }
    else
    {
        String authFailedStr = request.getParameter("authFailed");
        boolean authFailed = false;
        if ("true".equals(authFailedStr))
        {
            authFailed = true;
        }
        else
        {
            authFailed = false;
        }

        if (!authFailed)
        {
            //Get page parameters:
            String userId="";
            if(request.getParameter("userId") != null)
            {
                userId = request.getParameter("userId");
            }
            String pwd="";
            if(request.getParameter("pwd") != null)
            {
                pwd = request.getParameter("pwd");
            }

            if(userId != null && (!"".equals(userId.trim()))
                && pwd != null && (!"".equals(pwd.trim()))
            {
                try {
                    HttpSession userSession = request.getSession(true);

                    IWorkflowServiceClient wfSvcClient =
                        WorkflowServiceClientFactory.getWorkflowServiceClient
                            (WorkflowServiceClientFactory.REMOTE_CLIENT);
                    IWorkflowContext wfCtx =
                        wfSvcClient.getTaskQueryService().authenticate(userId, pwd, null);
                    httpSession.setAttribute("workflowContext", wfCtx);
                    response.sendRedirect(redirectPrefix + "statusPage.jsp");
                }
                catch (Exception e)

```

```

        {
            String worklistServiceError = e.getMessage();
            response.sendRedirect(redirectPrefix + "login.jsp?authFailed=true");
            out.println("error is " + worklistServiceError);
        }
    } else
    {
        out.println("Authentication failed");
    }
}
}
%>

<form action='<%= request.getRequestURI() %>' method="post">
<div style="width:100%">
<table cellpadding="2" cellspacing="3" border="0" width="30%" align="center">
<tr>
<td>Username
</td>
<td>
<input type="text" name="userId"/>
</td>
</tr>
<tr>
<td>Password
</td>
<td>
<input type="password" name="pwd"/>
</td>
</tr>
<tr>
<td>
<input type="submit" value="Submit"/>
</td>
</tr>
</table>
</form>
</div>
</center>
</body>
</html>

```

2. Query tasks using the queryTask API from TaskQueryService.

```

//add list of attributes to be queried from the task
List displayColumns = new ArrayList();
displayColumns.add("TASKNUMBER");
displayColumns.add("TITLE");
displayColumns.add("PRIORITY");
displayColumns.add("STATE");
displayColumns.add("UPDATEDDATE");
displayColumns.add("UPDATEDBY");
displayColumns.add("CREATOR");
displayColumns.add("OUTCOME");
displayColumns.add("CREATEDDATE");
displayColumns.add("ASSIGNEEUSERS");
displayColumns.add("ASSIGNEEGROUPS");
// get the list of tasks
List tasks = wfSvcClient.getTaskQueryService().queryTasks

```

```

        (wfCtx,
        displayColumns,
        null,
        ITaskQueryService.AssignmentFilter.MY_AND_GROUP,
        null,
        null,
        null,
        0,
        0);
    // create listing page by using above tasks
    //add href links to title to display details of the task by passing taskId
    as input parameter
    Use getTaskDetailsById(IWorkflowContext wftx, String taskId);

```

This is Step 4 in [Section 28.1, "Introduction to Building Clients for Workflow Services."](#)

The `statusPage.jsp` file of `HelpDeskRequest` is used to display the status of help desk requests. [Example 28–10](#) shows the `statusPage.jsp` example code.

Example 28–10 `statusPage.jsp`

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<%@ page import="oracle.tip.pc.services.identity.BPMAuthorizationService,
    oracle.bpel.services.workflow.verification.IWorkflowContext,
    oracle.tip.pc.services.common.ServiceFactory,
    oracle.bpel.services.workflow.client.IWorkflowServiceClient,
    oracle.bpel.services.workflow.client.WorkflowServiceClientFactory,
    oracle.bpel.services.workflow.query.ITaskQueryService,
    oracle.bpel.services.workflow.task.model.Task,
    oracle.bpel.services.workflow.task.model.IdentityType,
    oracle.tip.pc.services.identity.BPMUser,
    java.util.List,
    java.util.Calendar,
    java.text.SimpleDateFormat,
    java.util.ArrayList"%>
<%@ page contentType="text/html; charset=UTF-8"%>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8"/>
<title>RequestPage</title>
<style TYPE="text/css">
    Body, Form, Table, Textarea, Select, Input, Option
    {
        font-family : tahoma, verdana, arial, helvetica, sans-serif;
        font-size : 9pt;
    }
    table.banner
    {
        background-color: #eaeff5;
    }
    tr.userInfo
    {
        background-color: #eaeff5;
    }
    tr.problemInfo
    {
        background-color: #87b4d9;
    }
</style>

```

```

</head>
<body bgcolor="White">
<%
    HttpSession httpSession = request.getSession(false);
    httpSession.setAttribute("pageType", "STATUSPAGE");
%>
<table bordercolor="#eaeff5" border="4" width="100%">
  <tr><td> <jsp:include page="banner.jsp" /> </td></tr>
</table>
<%
    BPMUser bpmUser = null;
    String redirectPrefix = request.getContextPath() + "/";
    IWorkflowContext ctx = null;
    if (httpSession != null) {

        ctx = (IWorkflowContext) httpSession.getAttribute("workflowContext");
        if (ctx != null) {
            bpmUser = getAuthorizationService(ctx.getIdentityContext()).
                lookupUser(ctx.getUser());
        }
        else
        {
            response.sendRedirect(redirectPrefix + "login.jsp");
            return;
        }
    }
    else
    {
        response.sendRedirect(redirectPrefix + "login.jsp");
        return;
    }
    if(bpmUser == null)
    {
        response.sendRedirect(redirectPrefix + "login.jsp");
        return;
    }
    String status = (String)httpSession.getAttribute("requeststatus");
    if(status != null && !status.equals(""))
    {
%>
        <p></p>
        <div style="text-align:left;color:red" >
            <%= status %>
        </div>
<%
    }
    httpSession.setAttribute("requeststatus",null);
    IWorkflowServiceClient wfSvcClient =
        WorkflowServiceClientFactory.getWorkflowServiceClient(
            WorkflowServiceClientFactory.REMOTE_CLIENT);
    List displayColumns = new ArrayList();
    displayColumns.add("TASKNUMBER");
    displayColumns.add("TITLE");
    displayColumns.add("PRIORITY");
    displayColumns.add("STATE");
    displayColumns.add("UPDATEDDATE");
    displayColumns.add("UPDATEDBY");
    displayColumns.add("CREATOR");
    displayColumns.add("OUTCOME");
    displayColumns.add("CREATEDDATE");

```

```

displayColumns.add("ASSIGNEEUSERS");
displayColumns.add("ASSIGNEEGROUPS");
List tasks = wfSvcClient.getTaskQueryService().queryTasks
    (ctx,
     displayColumns,
     null,
     ITaskQueryService.ASSIGNMENT_FILTER_CREATOR,
     null,
     null,
     null,
     0,
     0);
%>
<p></p>
<div style="text-align:left;color:green" >
  <b>
    Previous help desk request
  </b>
</div>
<p></p>
<div style="text-align:center" >
<table cellspacing="2" cellpadding="2" border="3" width="100%">
  <TR class="problemInfo">
    <TH>TaskNumber</TH>
    <TH>Title</TH>
    <TH>Priority</TH>
    <TH>CreatedDate</TH>
    <TH>Assignee(s)</TH>
    <TH>UpdatedDate</TH>
    <TH>UpdatedBy</TH>
    <TH>State</TH>
    <TH>Status</TH>
  </TR>
  <%
    SimpleDateFormat dflong = new SimpleDateFormat( "MM/dd/yy hh:mm a" );
    for(int i = 0 ; i < tasks.size() ; i ++)
    {
      Task task = (Task)tasks.get(i);
      int taskNumber = task.getSystemAttributes().getTaskNumber();
      String title = task.getTitle();
      int priority = task.getPriority();
      String assignee = getAssigneeString(task);
      Calendar createdDate = task.getSystemAttributes().getCreatedDate();
      Calendar updateDate = task.getSystemAttributes().getUpdatedDate();
      String updatedBy = task.getSystemAttributes().getUpdatedBy().getId();
      String state = task.getSystemAttributes().getState();
      String outcome = task.getSystemAttributes().getOutcome();
      if(outcome == null) outcome = "";
      String titleLink = "http://" + request.getServerName() +
        ":" + request.getServerPort() +
        "/integration/worklistapp/TaskDetails?taskId=" +
        task.getSystemAttributes().getTaskId();
    %>
    <tr class="userInfo">
      <td><%=taskNumber%></td>
      <td><a href="<%=titleLink%>" target="_blank"><%=title%></a></td>
      <td><%=priority%></td>
      <td><%=dflong.format(createdDate.getTime())%></td>
      <td><%=assignee%></td>
      <td><%=dflong.format(updateDate.getTime())%></td>

```

```

        <td><%=updatedBy%></td>
        <td><%=state%></td>
        <td><%=outcome%></td>
    <tr>
    <%
    }
    %>
</table>
</div>
<%!
    private BPMAuthorizationService getAuthorizationService(String identityContext)
    {
        BPMAuthorizationService authorizationService =
ServiceFactory.getAuthorizationServiceInstance();
        if (identityContext != null)
            authorizationService = ServiceFactory.getAuthorizationServiceInstance(identityContext);

        return authorizationService;
    }
    private String getAssigneeString(Task task) throws Exception
    {
        List assignees = task.getSystemAttributes().getAssigneeUsers();
        StringBuffer buffer = null;
        for(int i = 0 ; i < assignees.size() ; i++)
        {
            IdentityType type = (IdentityType)assignees.get(i);
            String name = type.getId();
            if(buffer == null)
            {
                buffer = new StringBuffer();
            }
            else
            {
                buffer.append(",");
            }
            buffer.append(name).append(" (U)");
        }
        assignees = task.getSystemAttributes().getAssigneeGroups();
        for(int i = 0 ; i < assignees.size() ; i++)
        {
            IdentityType type = (IdentityType)assignees.get(i);
            String name = type.getId();
            if(buffer == null)
            {
                buffer = new StringBuffer();
            }
            else
            {
                buffer.append(",");
            }
            buffer.append(name).append(" (G)");
        }
        if(buffer == null)
        {
            return "";
        }
        else
        {
            return buffer.toString();
        }
    }

```

```
    }  
    %>  
</body>  
</html>
```

Introduction to Human Workflow Services

This chapter describes how the human workflow services are used. These services perform a variety of operations in the life cycle of a task.

This appendix includes the following sections:

- [Section 29.1, "Introduction to Human Workflow Services"](#)
- [Section 29.2, "Notifications from Human Workflow"](#)
- [Section 29.3, "Assignment Service Configuration"](#)
- [Section 29.4, "Class Loading for Callbacks and Resource Bundles"](#)
- [Section 29.5, "Resource Bundles in Workflow Services"](#)
- [Section 29.6, "Introduction to Human Workflow Client Integration with Oracle WebLogic Server Services"](#)
- [Section 29.7, "Database Views for Oracle Workflow"](#)

29.1 Introduction to Human Workflow Services

This section describes the responsibilities of the following human workflow services.

- Task service
- Task query service
- Identity service
- Task metadata service
- User metadata service
- Task report service
- Runtime config service
- Evidence store service

29.1.1 Enterprise JavaBeans, SOAP, and Java Support for the Human Workflow Services

[Table 29–1](#) lists the type of Simple Object Access Protocol (SOAP), Enterprise JavaBeans, and Java support provided for the task services. Most human workflow services are accessible through SOAP and local and remote Enterprise JavaBeans APIs. You can use these services directly by using appropriate client proxies. Additionally, the client libraries are provided to abstract out the protocol details and provide a common interface for all transports.

Table 29–1 Enterprise JavaBeans, SOAP, and Java Support

Service Name	Supports SOAP Web Services	Supports Remote Enterprise JavaBeans	Supports Local Enterprise JavaBeans
Task Service: Provides task state management and persistence of tasks. In addition to these services, the task service exposes operations to update a task, complete a task, escalate and reassign tasks, and so on.	Yes	Yes	Yes
Task Query Service: Queries tasks for a user based on a variety of search criterion such as keyword, category, status, business process, attribute values, history information of a task, and so on.	Yes	Yes	Yes
Task Metadata Service: Exposes operations to retrieve metadata information related to a task.	Yes	Yes	Yes
Task Reports Service: Provides workflow report details.	Yes	Yes	Yes
User Metadata Service: Manages metadata related to workflow users, such as user work queues, preferences, vacation, and delegation rules.	Yes	Yes	Yes
Runtime Config Service: Provides methods for managing metadata used in the task service runtime environment.	Yes	Yes	Yes
Evidence Store Service: Supports storage and nonrepudiation of digitally-signed workflow tasks.	Yes	Yes	Yes
Identity Service: Enables authentication of users and the lookup of user properties, roles, group memberships, and privileges.	Yes	No	No

Table 29–2 lists the location for the SOAP Web Services Description Language (WSDL) file for each task service.

Table 29–2 SOAP WSDL Location for the Task Services

Service name	SOAP WSDL location
Task Service	<code>http://host:port/integration/services/TaskService/TaskServicePort?WSDL</code>
Task Query Service	<code>http://host:port/integration/services/TaskQueryService/TaskQueryService?WSDL</code>
Identity Service	<code>http://host:port/integration/services/IdentityService/configuration?WSDL</code> <code>http://host:port/integration/services/IdentityService/identity?WSDL</code>
Task Metadata Service	<code>http://host:port/integration/services/TaskMetadataService/TaskMetadataServicePort?WSDL</code>
User Metadata Service	<code>http://host:port/integration/services/UserMetadataService/UserMetadataService?WSDL</code>

Table 29–2 (Cont.) SOAP WSDL Location for the Task Services

Service name	SOAP WSDL location
Task Report Service	<code>http://host:port/integration/services/TaskReportService/TaskReportServicePort?WSDL</code>
Runtime Config Service	<code>http://host:port/integration/services/RuntimeConfigService/RuntimeConfigService?WSDL</code>
Evidence Store Service	<code>http://host:port/integration/services/EvidenceService/EvidenceService?WSDL</code>

Table 29–3 lists the JNDI names for the different Enterprise JavaBeans.

Table 29–3 JNDI Names for the Different Enterprise JavaBeans

Service name	JNDI Names for the Different Enterprise JavaBeans
Task Service	<code>ejb/bpel/services/workflow/TaskServiceBean</code>
Task Service Enterprise JavaBeans participating in client transaction	<code>ejb/bpel/services/workflow/TaskServiceGlobalTransactionBean</code>
Task Metadata Service	<code>ejb/bpel/services/workflow/TaskMetadataServiceBean</code>
Task Query Service	<code>ejb/bpel/services/workflow/TaskQueryService</code>
User Metadata Service	<code>ejb/bpel/services/workflow/UserMetadataService</code>
Runtime Config Service	<code>ejb/bpel/services/workflow/RuntimeConfigService</code>
Task Report Service	<code>ejb/bpel/services/workflow/TaskReportServiceBean</code>
Task Evidence Service	<code>ejb/bpel/services/workflow/TaskEvidenceServiceBean</code>

For more information about the client library for worklist services, see [Chapter 28, "Building a Custom Worklist Client"](#)

29.1.2 Security Model for Services

With the exception of the identity service, all services that use the above-mentioned APIs (SOAP, remote Enterprise JavaBeans, local Enterprise JavaBeans, and Java WSIF) require authentication to be invoked. All the above channels support passing the user identity using the human workflow context. The human workflow context contains either of the following:

- Login and password
- Token

The task query service exposes the `authenticate` operation that takes the login and password and returns the human workflow context used for all services. Optionally, with each request, you can pass the human workflow context with the login and password.

The `authenticate` operation also supports the concept of creating the context on behalf of a user with the admin ID and admin password. This operation enables you to create the context for a logged-in user to the Oracle BPM Worklist if the password for that user is not available.

Oracle recommends that you get the workflow context one time and use it everywhere. There are performance implications for getting the workflow context for every request.

A realm is an identity service context from the identity configuration. The realm name can be null if the default configuration is used.

29.1.2.1 Limitation on Propagating Identity to Workflow Services when Using SOAP Web Services

Identity propagation is the replication of authenticated identities across multiple SOAP web services used to complete a single transaction. SOAP web services also support web service security. When web service security is used, the human workflow context does not need to be present in the SOAP input. The web service security can be configured from the Oracle Enterprise Manager Fusion Middleware Control Console.

Note: Human workflow SOAP clients have been enhanced to work with Security Assertion Markup Language (SAML) token-based identity propagation when the web service is secured.

29.1.2.2 Creating Human Workflow Context on Behalf of a User

The `authenticateOnBehalfOf` API method on the task query service can create the human workflow context on behalf of a user by passing the user ID and password of an admin user in the request. An admin user is a user with the `workflow.admin` privilege. This created context is as if it was created using the password on behalf of the user.

This is useful for environments in which a back-end system acts on workflow tasks while users act in their own system. There is no direct interaction with workflow services; the system can use the on-behalf-of-user login to get a context for the user.

In [Example 29-1](#), the human workflow context is created for user `jcooper`.

Example 29-1 Human Workflow Context Creation

```
String adminUser = "...."
String adminPassword = "...."
String realm = "...."

IWorkflowContext adminCtx =
taskQueryService.authenticate(user,password.toCharArray(),realm);

IWorkflowContext behalfOfCtx =
taskQueryService.authenticateOnBehalfOf(adminCtx,"jcooper");
```

29.1.3 Task Service

The task service exposes operations to act on tasks. [Table 29-4](#) describes the operations of the task service. Package `oracle.bpel.services.workflow.task` corresponds to the task service.

Table 29-4 Task Service Methods

Method	Description
<code>acquireTask</code>	Acquire a task.
<code>acquireTasks</code>	Acquire a set of tasks.
<code>addAttachment</code>	Add an attachment to a task.
<code>addComment</code>	Add a comment to a task.

Table 29–4 (Cont.) Task Service Methods

Method	Description
<code>createToDoTask</code>	Create a to-do task.
<code>delegateTask</code>	Delegate a task to a different user. Both the current assignee and the user to whom the task is delegated can view and act on the task.
<code>delegateTasks</code>	Delegate a list of tasks to a different user. Both the current assignee and the user to whom the list of tasks is delegated can view and act on the tasks.
<code>deleteTask</code>	Perform a logical deletion of a task. The task still exists in the database.
<code>deleteTasks</code>	Perform a logical deletion of a list of tasks. The tasks still exist in the database.
<code>errorTask</code>	Cause the task to error. This operation is typically used by the error assignee.
<code>escalateTask</code>	Escalate a task. The default escalation is to the manager of the current user. This can be overridden using escalation functions.
<code>escalateTasks</code>	Escalate tasks in bulk. The default escalation is to the manager of the current user. This can be overridden using escalation functions.
<code>getApprovers</code>	Get the previous approvers of a task.
<code>getFutureParticipants</code>	Get the future participants of a task. The future participants are returned in the form of a routing slip that contains simple participants (participant node and parallel nodes that contain routing slips in them).
<code>getUsersToRequestInfoForTask</code>	Get the users from whom a request for information can be requested.
<code>initiateTask</code>	Initiate a task.
<code>mergeAndUpdateTask</code>	Merge and update a task. Use this operation when a partial task should be updated. A partial task is one in which not all the task attributes are present. In this partial task, only the following task attributes are interpreted: <ul style="list-style-type: none"> ■ Task payload ■ Comments ■ Task state ■ Task outcome
<code>overrideRoutingSlip</code>	Override the routing slip of a task instance with a new routing slip. The current task assignment is nullified and the new routing slip is interpreted as its task is initiated.
<code>purgeTask</code>	Remove a task from the persistent store.
<code>purgeTasks</code>	Remove a list of tasks from the persistent store.
<code>pushBackTask</code>	Push back a task to the previous approver or original assignees. The original assignees do not need to be the approver as they may have reassigned the task, escalated the task, and so on. The property <code>pushbackAssignee</code> in <code>workflow-config.xml</code> controls whether the task is pushed back to the original assignees or the approvers.
<code>reassignTask</code>	Reassign a task.
<code>reassignTasks</code>	Reassign tasks in bulk.

Table 29–4 (Cont.) Task Service Methods

Method	Description
<code>reinitiateTask</code>	Reinitiate a task. Reinitiating a task causes a previously completed task to be carried forward so that the history, comments, and attachments are carried forward in a new task.
<code>releaseTask</code>	Release a previously acquired task.
<code>releaseTasks</code>	Release a set of previously acquired tasks.
<code>removeAttachment</code>	Remove a task attachment.
<code>renewTask</code>	Renew a task to extend the time it takes to expire.
<code>requestInfoForTask</code>	Request information for a task.
<code>requestInfoForTaskWithReapproval</code>	Request information for a task with reapproval. For example, assume <code>jcooper</code> created a task and <code>jstein</code> and <code>wfaulk</code> approved the task in the same order. When the next approver, <code>cdickens</code> , requests information with reapproval from <code>jcooper</code> , and <code>jcooper</code> submits the information, <code>jstein</code> and <code>wfaulk</code> approve the task before it comes to <code>cdickens</code> . If <code>cdickens</code> requests information with reapproval from <code>jstein</code> , and <code>jstein</code> submits the information, <code>wfaulk</code> approves the task before it comes to <code>cdickens</code> .
<code>resumeTask</code>	Resume a task. Operations can only be performed by the task owners (or users with the <code>BPMWorkflowSuspend</code> privilege) to remove the hold on a workflow. After a human workflow is resumed, actions can be performed on the task.
<code>resumeTasks</code>	Resume a set of tasks.
<code>routeTask</code>	Allow a user to route the task in an ad hoc fashion to the next user(s) who must review the task. The user can specify to route the tasks in sequential, parallel, or simple assignment. Routing a task is permitted only when the human workflow permits ad hoc routing of the task.
<code>skipCurrentAssignment</code>	Skip the current assignment and move to the next assignment or pick the outcome as set by the previous approver if there are no more assignees.
<code>submitInfoForTask</code>	Submit information for a task. This action is typically performed after the user has made the necessary updates to the task or has added comments or attachments containing additional information.
<code>suspendTask</code>	Allow task owners (or users with the <code>BPMWorkflowSuspend</code> privilege) to put a human workflow on hold temporarily. In this case, task expiration and escalation do not apply until the workflow is resumed. No actions are permitted on a task that has been suspended (except resume and withdraw).
<code>suspendTasks</code>	Suspend a set of tasks.
<code>updateOutcomeOfTasks</code>	Update the outcome of a set of tasks.
<code>updateTask</code>	Update the task.
<code>updateTaskOutcome</code>	Update the task outcome.
<code>updateTaskOutcomeAndRoute</code>	Update the task outcome and route the task. Routing a task allows a user to route the task in an ad hoc fashion to the next user(s) who must review the task. The user can specify to route the tasks in serial, parallel, or single assignment. Routing a task is permitted only when the human workflow permits ad hoc routing of the task.

Table 29–4 (Cont.) Task Service Methods

Method	Description
<code>withdrawTask</code>	The creator of the task can withdraw any pending task if they are no longer interested in sending it further through the human workflow. A task owner can also withdraw a task on behalf of the creator. When a task is withdrawn, the business process is called back with the state attribute of the task set to <code>Withdrawn</code> .
<code>withdrawTasks</code>	Withdraw a set of tasks.

For more information, see the following:

- [Section 29.1.11, "Task Instance Attributes"](#)
- *Oracle Fusion Middleware Workflow Services Java API Reference for Oracle BPEL Process Manager*
- `Sample workflow-118-JavaSamples`, which demonstrates some APIs

29.1.4 Task Query Service

The task query service queries tasks based on a variety of search criterion such as keyword, category, status, business process, attribute values, history information of a task, and so on. [Table 29–5](#) describes the operations of the task query service, including how to use the service over SOAP. Package `oracle.bpel.services.workflow.query` corresponds to the task query service.

Table 29–5 Task Query Service Methods

Method	Description
<code>authenticate</code>	Authenticates a user with the identity authentication service and passes back a valid <code>IWorkflowContext</code> object.
<code>authenticateOnBehalfOf</code>	Optionally make authentication on behalf of another user.
<code>countTasks</code>	Counts the number of tasks that match the specified query criteria.
<code>countViewTasks</code>	Counts the number of tasks that match the query criteria of the specified view.
<code>createContext</code>	Creates a valid <code>IWorkflowContext</code> object from a preauthenticated HTTP request.
<code>doesTaskExist</code>	Checks to see if any tasks exist that match the specified query criteria.
<code>doesViewTaskExist</code>	Checks to see if any tasks exist match the query criteria of the specified view.
<code>getWorkflowContext</code>	Gets a human workflow context with the specified context token.
<code>destroyWorkflowContext</code>	Cleans up a human workflow context that is no longer needed. This method is typically used when a user logs out.
<code>getTaskDetailsById</code>	Gets the details of a specific task from the task's <code>taskId</code> property.
<code>getTaskDetailsByNumber</code>	Gets the details of a specific task from the task's <code>task number</code> property.
<code>getTaskHistory</code>	Gets a list of the task versions for the specified task ID.

Table 29–5 (Cont.) Task Query Service Methods

Method	Description
<code>getTaskSequence</code>	Gets the task sequence tree of a task whose ID is a task ID, for those type of sequence.
<code>getTaskVersionDetails</code>	Gets the specific task version details for the specified task ID and version number.
<code>queryAggregatedTasks</code>	Executes the specified query, and aggregates a count of the tasks returned by the query, grouped by the specified column.
<code>queryTaskErrors</code>	Returns a list of task error objects matching the specified predicate.
<code>queryTasks</code>	<p>Returns a list of tasks that match the specified filter conditions. Tasks are listed according to the ordering condition specified (if any). The entire list of tasks matching the criteria can be returned or clients can execute paging queries, in which only a specified number of tasks in the list are retrieved. The filter conditions are as follows:</p> <ul style="list-style-type: none"> ▪ <i>assignmentFilter</i>: Filters tasks according to whom the task is assigned, or who created the task. Possible values for the assignment filter are as follows: <ul style="list-style-type: none"> ADMIN: No filtering; returns all tasks regardless of assignment or creator. ALL: No filtering; returns all tasks regardless of assignment or creator. CREATOR: Returns tasks in which the context user is the creator. GROUP: Returns tasks that are assigned to a group, application role, or list of users of which the context user is a member. MY: Returns tasks that are assigned exclusively to the context user. MY_AND_GROUP: Returns tasks that are assigned exclusively to the context user, or to a group, application role, or list of users of which the context user is a member. OWNER: Returns tasks in which the context user is the task owner. PREVIOUS: Returns tasks the context user previously updated. REPORTEES: Returns tasks that are assigned to reportees of the context user. REVIEWER: Returns tasks for which the context user is a reviewer. ▪ <i>keywords</i>: An optional search string. This only returns tasks in which the string is contained in the task title, task identification key, or one of the task text flex fields. ▪ <i>predicate</i>: An optional <code>oracle.bpel.services.workflow.repos.Predicate</code> object that allows clients to specify complex, SQL-like query predicates.
<code>queryViewAggregatedTasks</code>	Executes the query as defined in the specified view, and aggregates the selected tasks according to the chart property defined in the view.

Table 29–5 (Cont.) Task Query Service Methods

Method	Description
queryViewTasks	Returns a list of tasks according to the criteria in the specified view. The entire list or paged list of tasks can be returned. Clients can specify additional filter and ordering criteria to those in the view.

For more information, see the following:

- [Section 29.1.11, "Task Instance Attributes"](#)
- *Oracle Fusion Middleware Workflow Services Java API Reference for Oracle BPEL Process Manager* in the documentation library
- Sample workflow-118-JavaSamples, which demonstrates some APIs

29.1.5 Identity Service

The identity service is a thin web service layer on top of the Oracle Application Server 11g security infrastructure, namely Oracle Identity Management and Oracle Platform Security Services (OPSS), or any custom user repository. The identity service enables authentication of users and the lookup of user properties, roles, group memberships, and privileges. Oracle Identity Management is the sole identity service provider for Oracle Application Server 11g. Oracle Identity Management handles all storage and retrieval of users and roles for various repositories, including XML, LDAP, and so on. More specifically, Oracle Identity Management provides the following features:

- All providers are supported through Oracle Identity Management. The OracleAS JAAS Provider (JAZN) and LDAP providers are no longer supported. The custom provider is deprecated and supported only for backward compatibility. All customization of providers is performed through the custom provider to Oracle Identity Management, through configuring Oracle Virtual Directory (OVD) as an LDAP provider to Oracle Identity Management, or through both. OVD aggregates data across various repositories.
- The OPSS layer is used, which includes the following:
 - Identity store
 - Policy store
 - Credential store
 - Framework

For more information, see *Oracle Fusion Middleware Security Guide*. All security configuration is done through the `jps-config.xml` file.

- All privileges are validated against permissions, as compared to actions in previous releases.
- The following set of application roles are defined. These roles are automatically defined in the `soa-infra` application of the OPSS policy store.
 - `SOAAdmin`: Grant this role to users who must perform administrative actions on any SOA module. This role is also granted the `BPMWorkflowAdmin` and `B2BAdmin` roles.
 - `BPMWorkflowAdmin`: Grant this role to users who must perform any workflow administrative action. This includes actions such as searching and acting on any task in the system, creating and modifying user and group rules,

performing application customization, and so on. This role is granted the `BPMWorkflowCustomize` role and the following permissions:

- * `workflow.mapping.protectedFlexField`
- * `workflow.admin.evidenceStore`
- * `workflow.admin`

- `BPMWorkflowCustomize`: Grant this role to business users who must perform flex field mapping to public flex fields. This role is also granted the `workflow.mapping.publicFlexField` permission.
- The following workflow permissions are defined:
 - `workflow.admin`: Controls who can perform administrative actions related to tasks, user and group rules, and customizations
 - `workflow.admin.evidenceStore`: Controls who can view and search evidence records related to digitally-signed tasks (tasks that require a signature with the use of digital certificates).
 - `workflow.mapping.publicFlexField`: Controls who can perform mapping of task payload attributes to public flex fields.
 - `workflow.mapping.protectedFlexField`: Controls who can perform mapping of task payload attributes to protected flex fields.

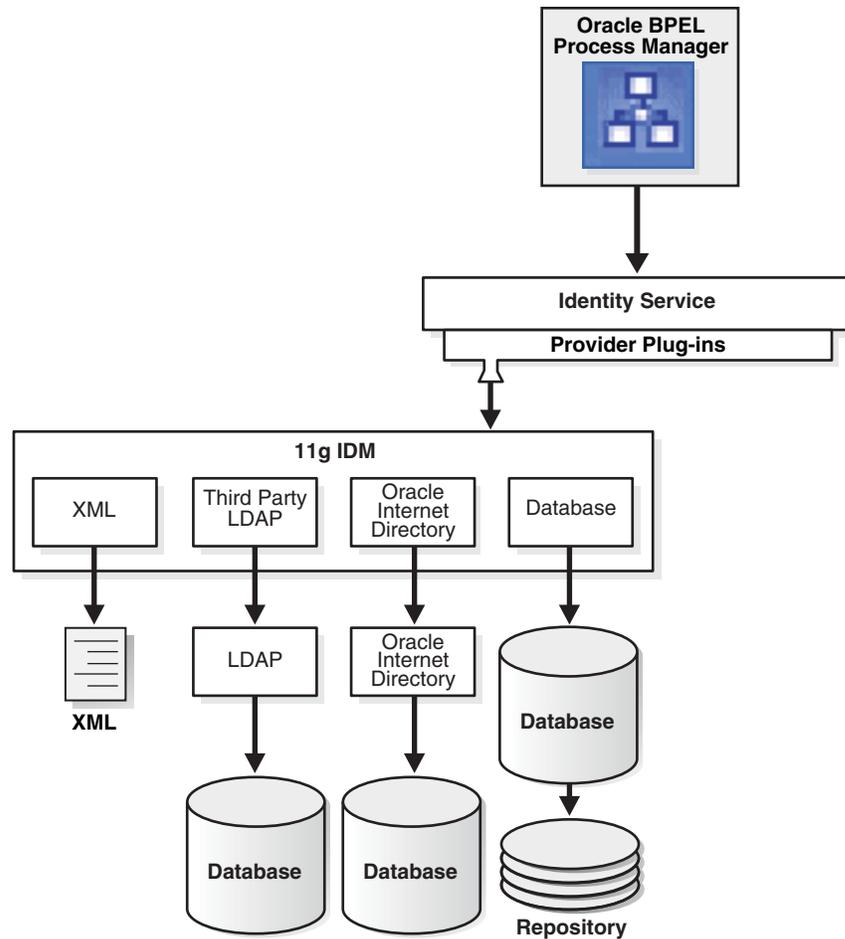
Note: You cannot specify multiple authentication providers for Oracle SOA Suite. This is because OPSS does not support multiple providers. The provider to use for human workflow authentication must be the first one listed in the order of authentication providers for Oracle SOA Suite.

For more information, see the following:

- *Oracle Fusion Middleware Security Guide* for details about OPSS
- *Oracle Fusion Middleware Application Developer's Guide for Oracle Identity Management* for details about Oracle Identity Management
- *Oracle Fusion Middleware Administrator's Guide for Oracle Virtual Directory* for details about OVD

29.1.5.1 Identity Service Providers

Oracle Identity Management is the only supported provider for release 11g, as shown in [Figure 29-1](#).

Figure 29–1 Identity Service Providers

29.1.5.1.1 Custom User Repository Plug-ins This mode enables you to plug in a non-LDAP-based user repository by registering a custom identity service provider. This mode is provided only for backward compatibility. The custom identity service plug-in must implement the `BPMIdentityService` interface (see the Javadoc). This `identityservice` class name must be registered in `workflow-identity-config.xml`.

29.1.6 Task Metadata Service

The task metadata service exposes operations to retrieve metadata information related to a task. [Table 29–6](#) describes these methods. Package `oracle.bpel.services.workflow.metadata` corresponds to the task metadata service.

Table 29–6 Task Metadata Service Methods

Method	Description
<code>getTaskMetadataByName space</code>	Get the <code>TaskMetadata</code> object that describes the human task service component with the specified task definition namespace and composite version.
<code>getOutcomes</code>	Get the permitted outcomes of a task. The outcomes are returned with their display values.

Table 29–6 (Cont.) Task Metadata Service Methods

Method	Description
<code>getResourceBundleInfo</code>	Get the resource bundle information of the task. The resource bundle information contains the location and the name of the bundle.
<code>getRestrictedActions</code>	Get the actions that are restricted for a particular task.
<code>getTaskAttributesForTaskDefinitions</code>	Get a list of <code>TaskAttribute</code> objects that describe standard task attributes and mapped flex-field columns that are common for the specified task definitions.
<code>getTaskAttributesForTaskNamespaces</code>	Get a list of <code>TaskAttribute</code> objects that describe standard task attributes and mapped flex field columns that are common for task definitions identified by the specified namespaces.
<code>getTaskAttributes</code>	Get the task message attributes.
<code>getTaskAttributesForTaskDefinition</code>	Get the message attributes for a particular task definition.
<code>getTaskDefinition</code>	Get the task definition associated with the task.
<code>getTaskDefinitionById</code>	Get the task definition by the task definition ID.
<code>getTaskDefinitionOutcome</code>	Get the outcomes given the task definition ID.
<code>getTaskDisplay</code>	Get the task display for a task.
<code>getTaskVisibilityRules</code>	Get the task visibility rules.
<code>getTaskDisplayRegion</code>	Get the task display region for a task.
<code>getVersionTrackedAttributes</code>	Get the task attributes that when changed cause a task version creation.
<code>listTaskMetadata</code>	List the task definitions in the system.

For more information, see *Oracle Fusion Middleware Workflow Services Java API Reference for Oracle BPEL Process Manager*.

29.1.7 User Metadata Service

The user metadata service provides methods for managing metadata specific to individual users and groups. It is used for getting and setting user worklist preferences, managing user custom views, and managing human workflow rules for users and groups.

For most methods in the user metadata service, the authenticated user can query and update their own user metadata. However, they cannot update metadata belonging to other users.

In the case of group metadata (for example, human workflow rules for groups), only a user designated as an owner of a group (or a user with the `workflow.admin` privilege) can query and update the metadata for that group. However, a user with the `workflow.admin` privilege can query and update metadata for any user or group.

[Table 29–7](#) describes the methods in the user metadata service. Package `oracle.bpel.services.workflow.user` corresponds to the user metadata service.

Table 29–7 User Metadata Service Methods

Method	Description
<code>createRule</code>	Creates a new rule.
<code>decreaseRulePriority</code>	Decreases the priority of a rule by one. This method does nothing if this rule has the lowest priority.
<code>deleteRule</code>	Deletes a rule.
<code>getVacationInfo</code>	Retrieves the date range (if any) during which a user is unavailable for the assignment of tasks.
<code>getRuleDetail</code>	Gets the details for a particular human workflow rule.
<code>getRuleList</code>	Retrieves a list of rules for a particular user or group.
<code>updateRule</code>	Updates an existing rule.
<code>increaseRulePriority</code>	Increases the priority of a rule by one. Rules for a user or group are maintained in an ordered list of priority. Higher priority rules (those closer to the head of the list) are executed before rules with lower priority. This method does nothing if this rule has the highest priority.
<code>getUserTaskViewList</code>	Gets a list of the user task views that the user owns.
<code>getGrantedTaskViewList</code>	Gets a list of user task views that have been granted to the user by other users. Users can use granted views for querying lists of tasks, but they cannot update the view definition.
<code>getStandardTaskViewList</code>	Gets a list of standard task views that ship with the human workflow service, and are available to all users.
<code>getUserTaskViewDetails</code>	Gets the details for a single view.
<code>createUserTaskView</code>	Creates a new user task view.
<code>updateUserTaskView</code>	Updates an existing user task view.
<code>deleteUserTaskView</code>	Deletes a user task view.
<code>updateGrantedTaskView</code>	Updates details of a view grant made to this user by another user. Updates are limited to hiding or unhiding the view grant (hiding a view means that the view is not listed in the main inbox page of Oracle BPM Worklist), and changing the name and description that the granted user sees for the view.
<code>getUserPreferences</code>	Gets a list of user preferences for the user. User preferences are simple name-value pairs of strings. User preferences are private to each user (but can still be queried and updated by a user with the <code>workflow.admin</code> privilege).
<code>setUserPreferences</code>	Sets the user preference values for the user. Any preferences that were previously stored and are not in the new list of user preferences are deleted.
<code>getPublicPreferences</code>	Gets a list of public preferences for the user. Public preferences are similar to user preferences, except any user can query them. However, only the user that owns the preferences, or a user with the <code>workflow.admin</code> privilege, can update them. Public preferences are useful for storing application-wide preferences (preferences can be stored under a dummy user name, such as <code>MyAppPrefs</code>).
<code>setPublicPreferences</code>	Sets the public preferences for the user.

Table 29–7 (Cont.) User Metadata Service Methods

Method	Description
setVacationInfo	Sets a date range over which the user is unavailable for the assignment of tasks. (Dynamic assignment functions do not assign tasks to a user that is on vacation.)
getStandardTaskViewDetails	Gets the full details for a particular standard view, identified by its viewId.

For more information, see the following:

- [Chapter 27, "Using Oracle BPM Worklist"](#) for details about the rule configuration and user preference pages
- `Sample workflow-118-JavaSamples`, which demonstrates some APIs
- *Oracle Fusion Middleware Workflow Services Java API Reference for Oracle BPEL Process Manager*

29.1.8 Task Report Service

The task report service executes a report and receives the results. [Table 29–8](#) describes the method. Package `oracle.bpel.services.workflow.report` corresponds to the task report service. The standard reports shown in [Table 29–8](#) are available as part of installation.

Table 29–8 Task Report Service

Report	Description
Unattended tasks report	Provides an analysis of tasks assigned to users' groups or reportees' groups that require attention because they have not yet been acquired.
Tasks priority report	Provides an analysis of the number of tasks by priorities assigned to a user, reportees, or their groups.
Tasks cycle time report	Provides an analysis of time taken to complete tasks from assignment to completion based on users' groups or reportees' groups.
Tasks productivity report	Provides an analysis of tasks assigned and tasks completed in a given time period for a user, reportees, or their groups.
Tasks time distribution report	Provides an analysis of time taken to complete their part of the tasks for a given user, user's groups, or reportees in the given time period.

29.1.9 Runtime Config Service

The runtime config service provides methods for managing metadata used in the task service runtime environment. It principally supports the management of task payload flex field mappings and the URIs used for displaying task details.

The task object used by the task service contains many flex field attributes, which can be populated with information from the task payload. This allows the task payload information to be queried, displayed in task listings, and used in human workflow rules.

The runtime config service provides methods for querying and updating the URI used for displaying the task details of instances of a particular task definition in a client application. For any given task definition, multiple display URIs can be supported,

with different URIs being used for different applications. The method `getTaskDisplayInfo` can query the URIs for a particular task definition. The method `setTaskDisplayInfo` can define new URIs or update existing ones. Only users with the `workflow.admin` privilege can call `setTaskDisplayInfo`, but any authenticated user can call `getTaskDisplayInfo`.

The runtime config service allows administrators to create mappings between simple task payload attributes and these flex field attributes.

Only a user with the `workflow.mapping.publicFlexField` or `workflow.mapping.protectedFlexField` privilege can make updates to payload mappings for public flex fields. Only a user with the `workflow.mapping.protectedFlexField` privilege can make updates to payload mappings for protected flex fields. Any authenticated user can use the query methods in this service.

An administrator can create attribute labels for the various flex field attributes. These attribute labels provide a meaningful label for the attribute (for example, a label `Location` may be created for the flex field attribute `TextAttribute1`). A given flex field attribute may have multiple labels associated with it. This attribute label is what is displayed to users when displaying lists of attributes for a specific task in Oracle BPM Worklist. The attribute labels for a specific task type can be determined by calling the `getTaskAttributesForTaskDefinition` method on the task metadata service.

When defining attribute labels, the following fields are automatically populated by the service. You do not need to specify values for these attributes when creating or updating attribute labels:

- `Id`
- `CreatedDate`
- `WorkflowType`
- `Active`

Valid values for the task attribute field for public flex fields are as follows:

- `TextAttribute1` through `TextAttribute20`
- `FormAttribute1` through `FormAttribute10`
- `UrlAttribute1` through `UrlAttribute10`
- `DateAttribute1` through `DateAttribute10`
- `NumberAttribute1` through `NumberAttribute10`

Mappings can then be created between task payload fields and the attribute labels. For example, the payload field `customerLocation` can be mapped to the attribute label `Location`. Different task types can share the same attribute label. This allows payload attributes from different task types that have the same semantic meaning to be mapped to the same attribute label.

Note: Only payload fields that are simple XML types can be mapped.

The runtime config service also provides the following:

- Methods for querying the dynamic assignment functions supported by the server

- Methods for maintaining the task display URLs used for displaying the task details in the Oracle BPM Worklist and other applications
- Methods for getting the server HTTP and JNDI URLs

Table 29–9 describes the methods in the runtime config service. Package `oracle.bpel.services.workflow.runtimeconfig` corresponds to the runtime config service.

Table 29–9 Runtime Config Service

Method	Description
<code>CreateAttributeLabel</code>	Creates a new attribute label for a particular task flex field attribute.
<code>createPayloadMapping</code>	Creates a new mapping between an attribute label and a task payload field.
<code>DeleteAttributeLabel</code>	Deletes an existing attribute label.
<code>deletePayloadMapping</code>	Deletes an existing payload mapping.
<code>getAttributeLabelUses</code>	Gets a list of attribute labels (either all attribute labels or labels for a specific type of attribute) for which mapping (if any) the labels are currently used.
<code>getGroupDynamicAssignmentFunctions</code>	Returns a list of the dynamic assignment functions that can select a group that are implemented on this server.
<code>getTaskDisplayInfo</code>	Retrieves information relating to the URIs used for displaying task instances of a specific task definition.
<code>getTaskStatus</code>	Gets the status of a task instance corresponding to a particular task definition and composite instance.
<code>getUserDynamicAssignmentFunctions</code>	Returns a list of the dynamic assignment functions that can select a user that are implemented on this server.
<code>GetWorkflowPayloadMappings</code>	Gets a list of all the flex field mappings for a particular human workflow definition.
<code>setTaskDisplayInfo</code>	Sets information relating to the URIs to be used for displaying task instances of a specific task definition.
<code>updateAttributeLabel</code>	Updates an existing attribute label.

For more information, see the following:

- [Section 29.3.1, "Dynamic Assignment and Task Escalation Functions"](#) for additional details
- [Chapter 27, "Using Oracle BPM Worklist"](#) for details about flex field mapping
- `Sample workflow-118-JavaSamples`, which demonstrates some APIs.
- *Oracle Fusion Middleware Workflow Services Java API Reference for Oracle BPEL Process Manager*

29.1.9.1 Internationalization of Attribute Labels

Attribute labels provide a method of attaching a meaningful label to a task flex field attribute. It can be desirable to present attribute labels that are translated into the appropriate language for the locale of the user.

To use a custom resource bundle, place it at the location identified by the workflow configuration parameter `workflowCustomClasspathURL` (which can be a file or HTTP path).

This can be set in either of two places in Oracle Enterprise Manager Fusion Middleware Control Console:

- System MBean Browser page
- Workflow Task Service Properties page

For more information, see the workflow-110-workflowCustomizations sample, which describes how to use this parameter. Visit the following URL for details:

http://www.oracle.com/technology/sample_code/products/hwf

Entries for flex field attribute labels must be of the form:

```
FLEX_LABEL.[label name]=Label Display Name
```

For instance, the entry for a label named Location is:

```
FLEX_LABEL.Location=Location
```

Note that adding entries to these files for attribute labels is optional. If no entry is present in the file, the name of the attribute label as specified using the API is used instead.

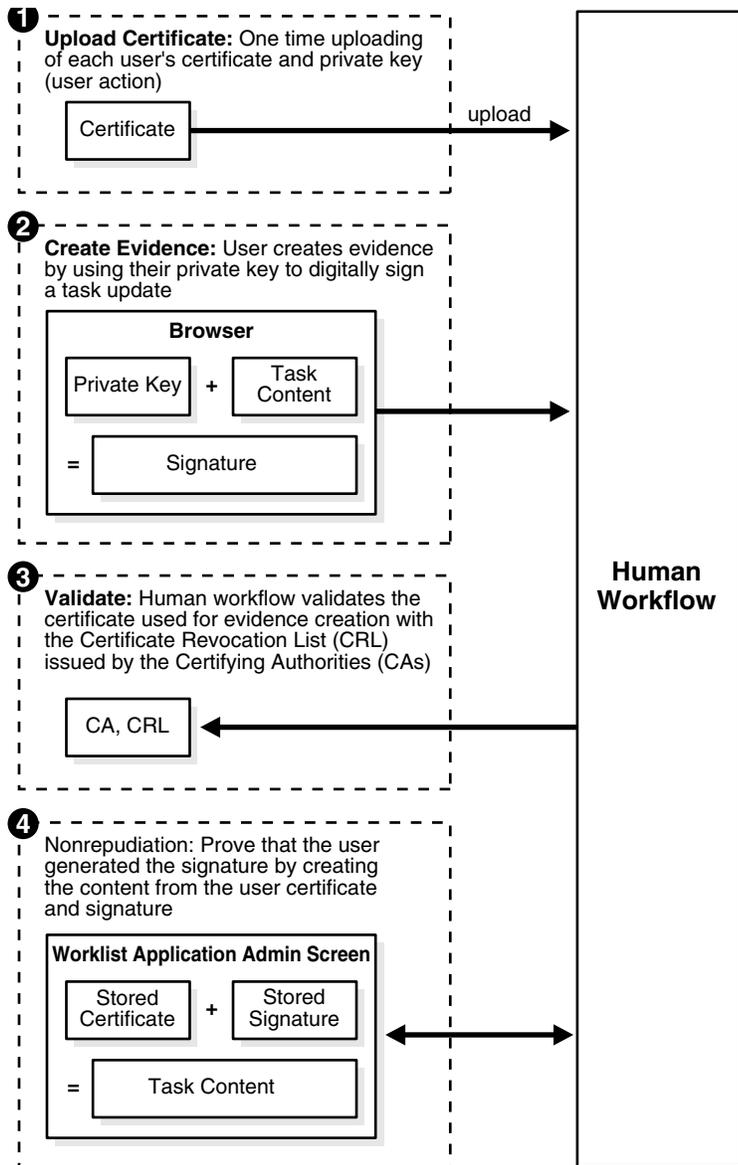
29.1.10 Evidence Store Service and Digital Signatures

The evidence store service is used for digital signature storage and nonrepudiation of digitally-signed human workflows. A digital signature is an electronic signature that authenticates the identity of a message sender or document signer. This ensures that the original content of the message or document sent is unchanged. Digital signatures are transportable, cannot be imitated by others, and are automatically time-stamped. The ability to ensure that the original signed message arrived means that the sender cannot repudiate it later. Digital signatures ensure that a human workflow document is authentic, has not been forged by another entity, has not been altered, and cannot be repudiated by the sender. A cryptographically-based digital signature is created when a public key algorithm signs a sender's message with a sender's private key.

During design time, signatures are enabled for the task. During runtime in the Oracle BPM Worklist, when a user approves or rejects the task, the web browser:

- Asks the user to choose the private key to use for signing.
- Generates a digital signature using the private key and task content provided by the Oracle BPM Worklist.

Figure 29–2 provides an example.

Figure 29–2 Digital Signature and Certificate**Notes:**

- The certificate refers to a Personal Information Exchange Syntax Standard (PFX) file that includes a certificate and a private key, and is protected by a simple text password. PFX specifies a portable format for storing or transporting a user's private keys, certificates, miscellaneous secrets, and so on.
- The possession of a private key that corresponds to the public key of a certificate is sufficient to sign the data, because the signature is verifiable through the public key in the certificate. However, no attempt is made to correlate the name of a user of a certificate with the person updating it. For example, user `jstein` can sign using the private key of user `cdickens` if `jstein` has that private key.

The following digital signature features are supported:

- PKCS7 signatures based on X.509 certificates
- Browser-based, digitally-signed content without attachments

29.1.10.1 Prerequisites

Prerequisites for using digital signatures and certificates are as follows:

- Users of the Oracle BPM Worklist must have certificates
- The administrator must specify the CAs and corresponding CRL URL whose certificates must be trusted. Users are expected to upload only certificates issued by these CAs. This is done by editing the System MBean Browser in Oracle Enterprise Manager Fusion Middleware Control Console.
 1. Log in to Oracle Enterprise Manager Fusion Middleware Control Console.
 2. In the navigator, expand the **SOA** folder.
 3. Right-click **soa-infra**, and select **Administration > System Mbean Browser**.
The System Mbean Browser displays on the right side of the page.
 4. Expand **Application Defined MBeans > oracle.as.soainfra.config > Server: server_name > WorkflowConfig > human-workflow**.
 5. Click the **Operations** tab on the right side of the page.
 6. Click **addTrustedCA**.
 7. Provide values for **caName** and **caURL**. For example, values provided for each invocation may look as shown in [Table 29–10](#).

Table 29–10 *caName and caURL Values*

caName	caURL
CN = Intg, OU =AppServ, O =Oracle, C = US	http://www.oracle.com/Integration%20CRL%20Data.crl
CN = Intg1, OU =AppServ, O =Oracle, C = US	http://www.oracleindia.in.com/Integration%20In.crl
CN = Intg2, OU =AppServ, O =Oracle, C = US	http://www.oracle.us.com/integration.crl

8. Click **Invoke**.

29.1.10.2 Interfaces and Methods

[Table 29–11](#) through [Table 29–14](#) describe the methods in the evidence store service. Package `oracle.bpel.services.security.evidence` corresponds to the evidence service.

Table 29–11 *ITaskEvidenceService Interface*

Method	Description
<code>createEvidence</code>	Creates evidence and stores it in the repository for nonrepudiation.

Table 29–11 (Cont.) ITaskEvidenceService Interface

Method	Description
getEvidence	Gets a list of evidence matching the given criteria. The result also depends on the privileges associated with the user querying the service. If the user has been granted the <code>workflow.admin.evidenceStore</code> permission (points to a location detailing how to grant the permission), all matching evidence is visible. Otherwise, only that evidence created by the user is visible.
uploadCertificate	Uploads certificates to be used later for signature verification. This is a prerequisite for creating evidence using a given certificate. A user can only upload their certificates.
updateEvidence	Updates the CRL verification part of the status. This includes verified time, status, and error messages, if any.
validateEvidenceSignature	Validates the evidence signature. This essentially performs a nonrepudiation check on the evidence. A value of <code>true</code> is returned if the signature is verified. Otherwise, <code>false</code> is returned.

Table 29–12 Evidence Interface

Method	Description
getCertificate	Gets the certificate used to sign this evidence.
getCreateDate	Gets the creation date of the evidence.
getErrorMessage	Gets the error message associated with the CRL validation.
getEvidenceId	Gets the unique identifier associated with the evidence.
getPlainText	Gets the content that was signed as part of this evidence.
getPolicy	Gets the signature policy of the evidence. This is either <code>PASSWORD</code> or <code>CERTIFICATE</code> .
getSignature	Gets the signature of this evidence.
getSignedDate	Gets the date on which the signature was created.
getStatus	Gets the CRL validation status. This can be one of the following: <ul style="list-style-type: none"> ■ <code>AVAILABLE</code>: The evidence is available for CRL validation. ■ <code>FAILURE</code>: CRL validation failed. ■ <code>SUCCESS</code>: CRL validation succeeded. ■ <code>UNAVAILABLE</code>: The CRL data could not be fetched. ■ <code>WAIT</code>: CRL validation is in progress.
getTaskId	Gets the unique identifier of the task with which this evidence is associated.
getTaskNumber	Gets the task number of the task with which this evidence is associated.
getTaskPriority	Gets the task priority of the task with which this evidence is associated.
getTaskStatus	Gets the task status of the task with which this evidence is associated.
getTaskSubStatus	Gets the task substatus of the task with which this evidence is associated.
getTaskTitle	Gets the title of the task with which this evidence is associated.

Table 29–12 (Cont.) Evidence Interface

Method	Description
getTaskVersion	Gets the version of the task with which this evidence is associated.
getVerifiedDate	Gets the date on which the CRL validation of the certificate used was performed.
getWorkflowType	Gets the workflow type of the task with which this evidence is associated. This is typically <code>BPELWF</code> .

Table 29–13 Certificate Interface

Method	Description
getCA	Gets the certificate issuer's distinguished name (DN).
getCertificate	Gets the certificate object that is abstracted by the interface.
getID	Gets the certificate's serial number.
getIdentityContext	Gets the identity context with which the uploader of this certificate is associated.
getUserName	Gets the user name with whom this certificate is associated.
isValid	Returns <code>true</code> if the certificate is still valid.

Table 29–14 Policy Type and Workflow Type Interface

Method	Description
fromValue	Constructs an object from the string representation.
value	Returns the string representation of this object.

For more information, see the following:

- [Section 25.3.10.6, "Specifying a Workflow Signature Policy"](#) for details about specifying digital signatures and digital certificates in the Human Task Editor
- [Chapter 26, "Designing Task Display Forms for Human Tasks"](#) for details about digitally signing a task action in the Oracle BPM Worklist

29.1.11 Task Instance Attributes

A task is work that must be done by a user. When you create a task, you assign humans to participate in and act upon the task. [Table 29–15](#) describes the task attributes that are commonly used and interpreted by applications.

Table 29–15 Task Attributes

Task Attribute Name	Description
task/applicationContext	The application with which any application roles associated with this task (assignees, owners, and so on) belong.
task/category	An optional category of the task.
task/creator	The name of the creator of this task.
task/dueDate	The due date for the task. This is used on to-do tasks.

Table 29–15 (Cont.) Task Attributes

Task Attribute Name	Description
task/identificationKey	An optional, custom, unique identifier for the task. This can be set as an additional unique identifier to the standard task ID and task number. This key can retrieve a task based on business object identifiers for which the task is created.
task/identityContext	The identity realm under which the users and groups are seeded. In a single realm environment, this defaults to the default realm.
task/ownerGroup	The group (if any) that owns this task instance. Task owners can be application roles, users, or groups. If the owner of the task is a group, this field is set.
task/ownerRole	The application role (if any) that owns this task instance. Task owners can be application roles, users, or groups. If the owner of the task is an application role, this field is set.
task/ownerUser	The user (if any) that owns this task instance. Task owners can be application roles, users, or groups. If the owner of the task is a user, this field is set.
task/payload	The task payload that is captured as XML.
task/percentageComplete	The percentage of the task completed. This is used on to-do tasks.
task/priority	An integer number that defines the priority of this task. A lower number indicates a higher priority. The numbers 1 to 5 are typically used.
task/startDate	The start date for the task. This is used on to-do tasks.
task/subCategory	An optional subcategory of the task.
task/taskDefinitionId	The task definition ID that binds the task to the task metadata. At task initiation time, this can be either the compositeDN/componentName string or the targetNamespace in the .task file. If the later is used, the active version matching the targetNamespace is used.
task/taskDisplayUrl	The URL to use to display the details for this task.
task/title	The title of the task.

Table 29–16 lists the attributes that capture process metadata information.

Table 29–16 Attributes Capturing Process Metadata Information

Attribute	Description
task/processInfo/domain	The domain to which the composite that contains the task component that defines this task instance belongs.
task/sca/applicationName	The application that is deployed.
task/sca/componentName	The name of the task component that defines this task instance.
task/sca/compositeDN	A unique name for the particular deployment of the composite that contains the task component that defines this task instance.
task/sca/compositeInstanceId	The composite instance ID.
task/sca/compositeName	The name of the composite that contains the task component that defines this task instance.

Table 29–16 (Cont.) Attributes Capturing Process Metadata Information

Attribute	Description
task/sca/compositeVersion	The version of the composite that contains the task component that defines this task instance.

Table 29–17 lists the attachment-related attributes.

Table 29–17 Attachment-related attributes

Attribute	Description
task/attachment/content	The attachment content.
task/attachment/mimeType	The Multipurpose Internet Mail Extension (MIME) type of the attachment.
task/attachment/name	The name of the attachment.
task/attachment/updatedBy	The user who updated the attachment.
task/attachment/updatedDate	The date on which the attachment was updated.
task/attachment/URI	The URI if the attachment is URI-based.

Table 29–18 lists the comment-related attributes.

Table 29–18 Comment-related Attributes

Attribute	Description
task/userComment/comment	The user comment.
task/userComment/updatedBy	The user who added the comment.
task/userComment/updatedDate	The date on which the comment was added.

Table 29–19 lists the attributes manipulated by the workflow services system.

Table 29–19 Attributes Manipulated by the Workflow Services System

Attribute	Description
task/systemAttributes/acquiredBy	If a task is assigned to a group, application role, or to multiple users, and then claimed by a user, this field is set to the name of the user who claimed the task.
task/systemAttributes/approvers	The IDs of users who performed custom actions on the task.
task/systemAttributes/assignedDate	The date that this task was assigned.
task/systemAttributes/assignees	The current task assignees (maybe users, groups, or application roles).
task/systemAttributes/createdDate	The date the task instance was created.
task/systemAttributes/customActions	The custom actions that can be performed on the task.
task/systemAttributes/endDate	The end date for the task. This is used on to-do tasks.

Table 29–19 (Cont.) Attributes Manipulated by the Workflow Services System

Attribute	Description
task/systemAttributes/expirationDate	The date on which the task instance expires.
task/systemAttributes/fromUser	The user who previously acted on the task.
task/systemAttributes/hasSubTasks	If true, there are subtasks.
task/systemAttributes/isGroup	If true, the task is assigned to a group.
task/systemAttributes/originalAssigneeUser	If a user delegates a task to another user, this field is populated with the name of the user who delegated the task.
task/systemAttributes/outcome	The outcome of the task (for example, approved or rejected). This is only set on completed task instances.
task/systemAttributes/parentTaskId	This is only set on reinitiated tasks (the task ID of the previous task that is being reinitiated).
task/systemAttributes/parentTaskVersion	This only set on a subtask. This refers to the version of the parent task.
task/systemAttributes/participantName	The logical name of the participant as modeled from Oracle JDeveloper.
task/systemAttributes/reviewers	The reviewers of the task. This can be a user, group, or application role.
task/systemAttributes/rootTaskId	The ID of the root task. This is equal to the task ID for the root task.
task/systemAttributes/stage	The stage name that is being executed.
task/systemAttributes/state	The current state of the task instance.
task/systemAttributes/substate	The current substate of the task.
task/systemAttributes/subTaskGroupId	A unique ID that is set on a subtask. This same ID is set on the parent task's taskGroupId. This is required to identify which subtasks were created at which time.
task/systemAttributes/systemActions	The system actions (such as reassign, escalate, and so on) that can be performed on a task.
task/systemAttributes/taskDefinitionName	The name of the task component that defines this task instance.
task/systemAttributes/taskGroupId	This only sets a subtask. This is the ID of the immediate parent task.
task/systemAttributes/taskGroupId	A unique ID that is set on the parent task. This same ID is set on the subtask's subTaskGroupId. This is required to identify which subtasks were created at which time.
task/systemAttributes/taskId	The unique ID of the task.
task/systemAttributes/taskNamespace	A namespace that uniquely defines all versions of the task component that defines this task instance. Different versions of the same task component can have the same namespace, but no two task components can have the same namespace.
task/systemAttributes/taskNumber	An integer number that uniquely identifies this task instance.

Table 29–19 (Cont.) Attributes Manipulated by the Workflow Services System

Attribute	Description
task/systemAttributes/updatedBy	The user who last updated the task.
task/systemAttributes/updatedAt	The date this instance was last updated.
task/systemAttributes/version	The version of the task.
task/systemAttributes/versionReason	The reason the version was created.
task/systemAttributes/workflowPattern	The pattern that is being executed (for example, parallel, serial, FYI, or single).

Table 29–20 lists the flex field attributes.

Table 29–20 Flex Field Attributes

Attribute	Description
task/systemMessageAttributes/*	The flex fields.

29.2 Notifications from Human Workflow

Notifications are sent to alert users of changes to the state of a task. Notifications can be sent through any of the following channels: email, telephone voice message, instant messaging (IM), or short message service (SMS). Notifications can be sent from a human task in a BPEL process or from a BPEL process directly.

In releases before 11g, email notifications were sent through the human workflow email notification layer. Voice and SMS notifications were sent through Oracle's hosted notification service. IM notifications were not supported.

Starting with release 11g, the human workflow email notification layer works with Oracle User Messaging Service to alert users to changes in the state of a task. The Oracle User Messaging Service exposes operations that can be invoked from the BPEL business process or human task to send notifications through email, voice, IM, or SMS channels.

The Oracle User Messaging Service supports features such as:

- Sending and receiving messages and statuses
- Sending notifications to a specific address on a particular channel
- Sending notifications to a set of failover addresses

On application servers other than Oracle Fusion Middleware, the human workflow email notification layer can be used for email notifications.

For more information about configuring the Oracle User Messaging Service, see the following:

- [Chapter 16, "Using the Notification Service"](#)
- *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite* for instructions on configuring notification service delivery channels in Oracle Enterprise Manager Fusion Middleware Control Console

29.2.1 Contents of Notification

Each email notification can contain the following parts:

- The notification message
- The HTML content from Oracle BPM Worklist:

This is a read-only view of Oracle BPM Worklist on the task. For information on how you can configure email notifications to include the content from Oracle BPM Worklist, see [Section 26.7, "Creating an Email Notification."](#)
- Task attachments:

For notifications that include task attachments.
- Actionable links

Notifications through SMS, IM, and voice contain *only* the notification message.

The notification message is an XPath expression that can contain static text and dynamic values. In creating the messages, only the task BPEL variable is available for dynamic values. This restriction is because the messages are evaluated outside the context of the BPEL process. The payload in the task variable is also strongly typed to contain the type of the payload for XPath tree browsing. The XPath extension function `hwf:getNotificationProperty(propertyName)` is available to get properties for a particular notification. The function evaluates to corresponding values for each notification. The `propertyName` can be one of the following values:

- `recipient`
The recipient of the notification
- `recipientDisplay`
The display name of the recipient
- `taskAssignees`
The task assignees
- `taskAssigneesDisplay`
The display names of the task assignees
- `locale`
The locale of the recipient
- `taskId`
The ID of the task for which the notification is meant
- `taskNumber`
The number of the task for which the notification is meant
- `appLink`
The HTML link to the Oracle BPM Worklist task details page

[Example 29–2](#) demonstrates the use of `hwf:getNotificationProperty` and `hwf:getTaskResourceBundle` together:

Example 29–2 Use of `hwf:getNotificationProperty` and `hwf:getTaskResourceBundle`

```
concat('Dear ', hwf:getNotificationProperty('recipientDisplay'), ' Task ',
/task:task/task:systemAttributes/task:taskNumber, ' is assigned to you. ',
hwf:getTaskResourceBundleString(/task:task/task:systemAttributes/task:taskId,
```

```
'CONGRATULATIONS', hwf:getNotificationProperty('locale')))
```

This results in a message similar to the following:

```
Dear Cooper, James Task 1111 is assigned to you. Congratulations
```

29.2.2 Error Message Support

The human workflow email notification layer is automatically configured to warn an administrator about error occurrences in which intervention is required. Error notifications and error response messages are persisted.

You can view messages in Oracle Enterprise Manager Fusion Middleware Control Console.

For more information about viewing messages, see *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite*.

29.2.3 Reliability Support

The human workflow email notification layer works with the Oracle User Messaging Service to provide the following reliability support:

- Messages are not lost:
 - If the human workflow email notification layer fails after acknowledging receipt of a message from the human workflow.
 - If the human workflow email notification layer and Oracle User Messaging Service both fail before the Oracle User Messaging Service acknowledges receipt of a message from the human workflow.
 - If the Oracle User Messaging Service is down. Message delivery is retried until successful.
 - If a notification channel is down.
- Notifications that cannot be delivered are retried three times and the address is marked as invalid. The address is also added to the bad address list. If needed, you can manually remove these addresses from the bad address list in Oracle Enterprise Manager Fusion Middleware Control Console. Outgoing notifications are not resent until the address is corrected. To guard against any incorrect identification, the address is marked as invalid only for about an hour. No new notifications are sent in this time.
- Incoming notification responses from an address that has been identified as a spam source are ignored.
- Incoming notification messages are persisted.
- Incoming notification responses that indicate notification delivery failure (for example, an unknown host mail) are not ignored; instead corrective actions are automatically taken (for example, the bad address list is updated).
- Incoming notification responses can be configured to send acknowledgements indicating notification status to the sender.
- Validation of incoming notification responses is performed by correlating the incoming notification message with the outgoing notification message.

For more information about notifications, see the following:

- [Chapter 16, "Using the Notification Service"](#)

- *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite*

29.2.4 Management of Oracle Human Workflow Notification Service

An administrator can perform the following management tasks from Oracle Enterprise Manager Fusion Middleware Control Console:

- View failed notifications and erroneous incoming notification responses and take corrective actions.
- Perform corrective actions such as delete, resend, and edit on outgoing notifications and addresses.
- View bad emails and block email addresses for incoming notification responses.
- Manage the bad email address list.
- Access runtime data of failed notifications. You can purge this data when it is no longer needed.

For more information, see *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite*.

29.2.5 How to Configure the Notification Channel Preferences

To configure the notification channel preferences:

1. In Oracle JDeveloper, configure the notification service for email and other channels. See [Chapter 16, "Using the Notification Service"](#) for details.
2. Open the Human Task Editor in Oracle JDeveloper.

The notifications for a task can be configured during the creation of a task in the Human Task Editor. Notifications can be sent to different types of participants for different actions.

The actions for which a task notification can be sent are described in [Section 25.3.9.1, "Notifying Recipients of Changes to Task Status."](#)

Notifications can be sent to users involved in the task in various capacities. These users are described in [Section 25.3.9.1, "Notifying Recipients of Changes to Task Status."](#)

When the task is assigned to a group, each user in the group is sent a notification if no notification endpoint is available for the group.

For more information, see the following:

- *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite* for details about configuring the notification channel
 - [Section 25.3.9, "How to Specify Participant Notification Preferences"](#) to configure task notifications in the Human Task Editor
 - [Chapter 16, "Using the Notification Service"](#)
3. From the messaging server pages of Oracle Enterprise Manager Fusion Middleware Control Console, configure the appropriate channel (for example, email). See *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite* for details.
 4. From the Workflow Notification Properties pages of Oracle Enterprise Manager Fusion Middleware Control Console, configure the notification mode parameter for the notification service to either all channels or email.

By default, this value is set to **NONE**, meaning that no notifications are sent. The possible values are:

- **ALL**

The email, IM, SMS, and voice channels are configured and notification is sent through any channel.

- **EMAIL**

Only the email channel is configured for sending notification messages.

- **NONE**

No channel is configured for sending notification messages. This is the default setting.

29.2.6 How to Configure Notification Messages in Different Languages

A notification consists of four types of data generated from multiples sources and internationalized differently. However, for all internationalized notifications, the locale is obtained from the `BPMUser` object of the identity service.

- Prepackaged strings (action links, comments, Oracle BPM Worklist, and so on)

These strings are internationalized in the product as part of the following package:

```
oracle.bpel.services.workflow.resource
```

The user's locale is used to get the appropriate message.

- Task details attachment

The user's locale is used to retrieve the task details HTML content.

- Task outcome strings (approve, reject, and so on)

The resource bundle for outcomes is specified when the task definition is modeled in the **Advanced Settings** section of the Human Task Editor. The key to each of the outcomes in the resource bundle is the outcome name itself.

- Notification message

To configure notification messages in different languages:

1. Use one of the following methods to internationalize messages in the notification content:

- a. If you want to use values from the resource bundle specified during the task definition, then use the following XPath extension function:

```
hwf:getTaskResourceBundleString(taskId, key, locale?)
```

This function returns the internationalized string from the resource bundle specified in the task definition.

The locale of the notification recipient can be retrieved with the following function:

```
hwf:getNotificationProperty('locale')
```

The task ID corresponding to a notification can be retrieved with the following function:

```
hwf:getNotificationProperty('taskId')
```

- b. If a different resource bundle is used, then use the following XPath extension to retrieve localized messages:

```
orcl:get-localized-string()
```

For more information, see [Section 25.3.10.4, "Specifying Multilingual Settings."](#)

29.2.7 How to Send Actionable Messages

There are several methods for sending actionable messages. This section provides an overview of procedures.

Note: If digital signatures are enabled for a task, actionable emails are not sent during runtime. This is the case even if actionable emails are enabled during design-time.

29.2.7.1 How to Send Actionable Emails for Human Tasks

Task actions can be performed through email if the task is set up to enable actionable email (the same actions can also be performed from Oracle BPM Worklist). An actionable email account is the account in which task action-related emails are received and processed.

To send actionable emails for human tasks:

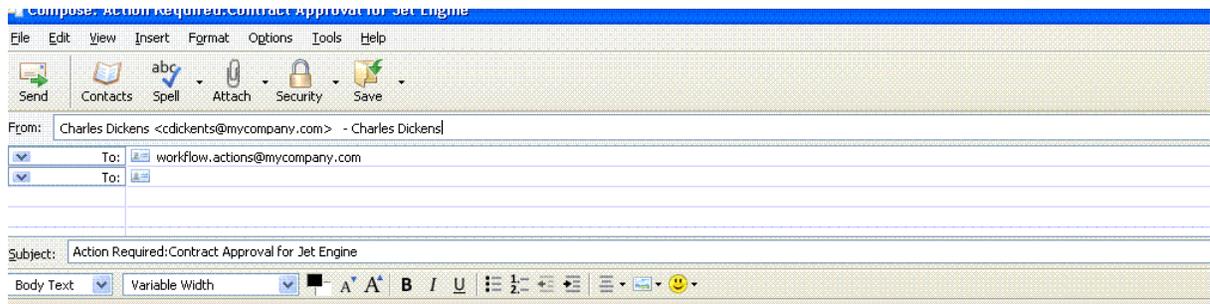
1. In the **Notification Settings** section of the Human Task Editor, select **Make notifications actionable** to make email notifications actionable. This action enables you to perform task actions through email.

If a notification is actionable, the email contains links for each of the custom outcomes.

2. If you want to send task attachments with the notification message, select **Send task attachments with email notifications**.

When an actionable email arrives, perform the following tasks.

3. Click the **Approve** link to invoke a new email window with approval data. [Figure 29–3](#) provides details.

Figure 29–3 Actionable Notifications

Add comments by editing the text between the brackets below.

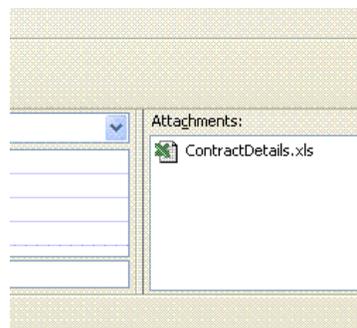
Comments: [This contract has been approved based on attached information.]

You can also add attachments to the task by attaching them to this email.

-----Do not edit below this line-----

Approve : [[NID]] : 8Scnzw12KyJYLkTs+bJJUazW51C96bzHKvEXkXJ08Ogk9opJ1QQQuDaNI7vq0Jq3D7DqYCRCS1b01GMXqHCrUjUYA0QvdsRxtYaXy
[[NID]]

4. Add comments in the comments section of the approval mail. For example:
This contract has been approved based on the attached information.
5. Add attachments as needed, as shown in [Figure 29–4](#).

Figure 29–4 Attachment to an Actionable Email

6. Do not change anything in the subject or the body in this email. If you change the content with the NID substrings, the email is not processed.
7. Click **Send**.
8. Set properties such as incoming server, outgoing mail server, outgoing username and password, and others from the Oracle User Messaging Service section of Oracle Enterprise Manager Fusion Middleware Control Console.
9. In the Workflow Notification Properties page of Oracle Enterprise Manager Fusion Middleware Control Console, set the notification mode to **ALL** or **EMAIL**.
10. In the Workflow Task Service Properties page of Oracle Enterprise Manager Fusion Middleware Control Console, set the actionable email account name.

29.2.8 How to Send Inbound and Outbound Attachments

If the include attachments flag is checked; only email is sent. The emails include all the task attachments as email attachments.

To send inbound and outbound attachments:

1. Select **Send task attachments with email notifications** in the **Notification Settings** section of the Human Task Editor.

In the actionable email reply, the user can add attachments in the email. These attachments are added as task attachments.

For more information, see [Section 25.3.9.6, "Making Email Messages Actionable."](#)

29.2.9 How to Send Inbound Comments

To send inbound comments:

1. Add comments in the actionable email reply between `Comments[[' and ']]`. Those contents are added as task comments. For example, `Comments[[looks good]]]`.

29.2.10 How to Send Secure Notifications

To send secure notifications:

1. Mark a notification as secure in the **Notification Settings** section of the Human Task Editor. This action enables a default notification message to be used. In this case, the notification message does not include the content of the task. Also, this notification is not actionable. The default notification message includes a link to the task in Oracle BPM Worklist. You must log in to see task details.

For more information, see [Section 25.3.9.5, "Securing Notifications to Exclude Details."](#)

29.2.11 How to Set Channels Used for Notifications

To set channels used for notifications:

1. Set up preferred notification channels by using the preferences user interface in Oracle BPM Worklist. The channel is dynamically determined by querying the user preference store before sending the notification. If the user preference is not specified, then the email channel is used.

For more information about the Oracle Delegated Administration Service, see *Oracle Fusion Middleware Guide to Delegated Administration for Oracle Identity Management*.

29.2.12 How to Send Reminders

Tasks can be configured to send reminders, which can be based on the time the task was assigned to a user or the expiration time of a task. The number of reminders and the interval between the reminders can also be configured. The message used for reminders is the message that is meant for ASSIGNEES when the task is marked as ASSIGNED.

To send reminders:

1. Set reminders in the **Notification Settings** section of the Human Task Editor. Reminder configuration involves the following parameters:
 - Recurrence:

The recurrence specifies the number of times reminders are sent. The possible values for recurrence are EVERY, NEVER, 0, 1, 2 ..., 10.

- **RelativeDate:**

The `RelativeDate` specifies if the reminder duration is computed relative to the assigned date or to the expiration date of the task. The possible values for the `relativeDate` are ASSIGNED, EXPIRATION, and BEFORE DUE DATE. The final value appears in Oracle JDeveloper if you modify the escalation and expiration policy of the task to use the option DUE DATE.

- **Duration:**

The duration from the `relativeDate` and the first reminder and each reminder since then. The data type of duration is `xsd:duration`, whose format is defined by ISO 8601 under the form `PnYnMnDTnHnMnS`. The capital letters are delimiters and can be omitted when the corresponding member is not used. Examples include `PT1004199059S`, `PT130S`, `PT2M10S`, `P1DT2S`, `-P1Y`, or `P1Y2M3DT5H20M30.123S`.

The following examples illustrate when reminders are sent:

- If the `relativeDate` is ASSIGNED, the recurrence is EVERY, the reminder duration is PT1D, and the task is assigned at 3/24/2005 10:00 AM, then reminders are sent at 3/25/2005 10:00 AM, 3/26/2005 10:00 AM, 3/27/2005 10:00 AM, and so on until the user acts on the task.
- If the `relativeDate` is EXPIRATION, the recurrence is 2, the reminder duration is PT1D, and the task expires at 3/26/2005 10:00 AM, then reminders are sent at 3/24/2005 10:00 AM and 3/25/2005 10:00 AM if the task was assigned before 3/24/2005 10:00 AM.
- If the `relativeDate` is EXPIRATION, the recurrence is 2, the reminder duration is PT1D, the task expires at 3/26/2005 10:00 AM, and the task was assigned at 3/24/2005 3:00 PM, then only one reminder is sent at 3/25/2005 10:00 AM.

For more information, see [Section 25.3.9.3, "Setting Up Reminders."](#)

29.2.13 How to Set Automatic Replies to Unprocessed Messages

The human workflow notification service sends you an automatic reply message when it cannot process an incoming message (due to system error, exception error, user error, and so on). You can modify the text for these messages in the global resource bundle.

For more information see [Section 29.5.2, "Global Resource Bundle – WorkflowLabels.properties."](#)

Example 29–3 *WorkflowLabels.properties*

```
# String to be prefixed to all auto reply messages
AUTO_REPLY_PREFIX_MESSAGE=Oracle Human Workflow Service
# String to be suffixed to all auto reply messages
AUTO_REPLY_SUFFIX_MESSAGE=This message was automatically generated by Human \
    Workflow Mailer. Do not reply to this mail.

# Message indicating closed status of a notified task
TaskClosed=You earlier received the notification shown below. That notification \
    is now closed, and no longer requires your response. You may \
    simply delete it along with this message.

# Message indicating that notification was "replied" to instead of "responded" by
```

```
# using the response link.
EMailRepliedNotification=The message you sent appeared to be a reply to a \
    notification. To respond to a notification, use the \
    response link that was included with your notification.

#
EMailUnSolicited= The message you sent did not appear to be in response to a \
    notification. If you are responding to a notification \
    Use the response link that was included with your notification.

EMailUnknownContent= The message you sent did not appear to be in response to a \
    notification. If you are responding to a notification, \
    Use the response link that was included with your notification.

ResponseNotProcessed=Your response to notification could not be processed. \
    Log in to worklist application for more details.

ResponseProcessed=Your response to notification was successfully processed.
```

29.2.14 How to Create Custom Notification Headers

Some task participants may have access to multiple notification channels. You can use custom notification headers to enable this type of participant to specify a single channel as the preferred channel on which to receive notifications.

To create custom notification headers:

1. In the **Custom Notification Headers** field of the **Notification Settings** section of the Human Task Editor, create custom notification headers that specify the preferred notification channel to use (such as voice, SMS, and so on). The human workflow email notification layer provides these header values to the rule-based notification service of the Oracle User Messaging Service for use.

For example, set the **Name** field to `deliveryType` and the **Value** field to `SMS`.

Note that the rule-based notification service is *only* used to identify the preferred notification channel to use. The address for the preferred channel is obtained from Oracle Identity Management. The notification message is created from the information provided by both services.

For more information, see the following:

- [Section 25.3.9.7, "Sending Task Attachments with Email Notifications"](#)
- [Chapter 42, "User Messaging Preferences"](#)

29.3 Assignment Service Configuration

This section describes how to configure the assignment service with dynamic assignment functions.

29.3.1 Dynamic Assignment and Task Escalation Functions

When tasks are assigned to a group, users in the group must typically claim a task to act on it. However, you can also automatically send work to users in the group by using various dispatching mechanisms. Automatic task dispatching is done through dynamic assignment functions. Dynamic assignment functions select a particular user or group from either a group, or from a list of users or groups. Several functions are automatically provided. However, you can also create your own functions and register

them with the workflow service. [Table 29–21](#) describes the three dynamic assignment functions.

Table 29–21 Dynamic Assignment Functions

Function	Type	Description
LEAST_BUSY	Dynamic assignment	Picks the user or group with the least number of tasks currently assigned to it.
MANAGERS_MANAGER	Task escalation	Picks the manager’s manager.
MOST_PRODUCTIVE	Dynamic assignment	Picks the user or group that has completed the most tasks over a certain time period (by default, the last seven days).
ROUND_ROBIN	Dynamic assignment	Picks each user or group in turn.

These functions all check a user’s vacation status. A user that is currently unavailable is not automatically assigned tasks.

These dynamic assignment functions can be called using the custom XPath functions in a BPEL process or task definition:

- `wfDynamicUserAssign`
- `wfDynamicGroupAssign`

These XPath functions must be called with at least two, and optionally more parameters:

- The name of the dynamic assignment function being called.
- The name of the group on which to execute the function (or a list of users or groups).
- (Optional) The identity realm to which the user or group belongs (the default value is the default identity realm).
- Additional optional parameters specific to the dynamic assignment function. In the case of the `MOST_PRODUCTIVE` assignment function, this is the length of time (in days) to calculate a user’s productivity. The two other functions do not use additional parameters.

In addition, human workflow rules created for a group can use dynamic assignment functions to select a member of that group for reassignment of a task.

In addition to these functions, a dynamic assignment framework is provided that enables you to implement and configure your own dynamic assignment functions.

29.3.1.1 How to Implement a Dynamic Assignment Function

Follow these procedures to implement your own dynamic assignment function.

To implement dynamic assignment functions:

1. Write a Java class that implements one or both of the following interfaces:

```
oracle.bpel.services.workflow.assignment.dynamic.
IDynamicUserAssignmentFunction
oracle.bpel.services.workflow.assignment.dynamic.
IDynamicGroupAssignmentFunction
```

2. If your dynamic assignment function selects users, implement the first interface. If it selects groups, implement the second interface. If it allows the selection of both users and groups, implement both interfaces.

The two interfaces above both extend the interface `oracle.bpel.services.workflow.assignment.dynamic.IDynamicAssignmentFunction`.

Your Java class should also implement the methods in that interface. These interfaces as shown in the Javadoc.

The dynamic assignment framework also provides the utility class `oracle.bpel.services.workflow.assignment.dynamic.DynamicAssignmentUtils`.

This class provides many methods that are useful when implementing dynamic assignment functions.

For information about the Javadoc for dynamic assignment interfaces and utilities, see `SOA_ORACLE_HOME\javadoc\soa-infra`.

29.3.1.2 How to Configure Dynamic Assignment Functions

Dynamic assignment functions are configured along with other human workflow configuration parameters in Oracle Enterprise Manager Fusion Middleware Control Console.

Each dynamic assignment has two mandatory parameters in this file, in the form of a `<function>` tag.

The function tag must contain two attributes:

- `name`: The name of the function
- `classpath`: The fully qualified class name of the class that implements the function.

In addition, each function can optionally have any number of properties. These properties are simple name-value pairs that are passed as initialization parameters to the function.

The property values specified in these tags are passed as a map (indexed by the value of the name attributes) to the `setInitParameters` method of the dynamic assignment functions.

Two of the functions have initialization parameters. These are:

- `ROUND_ROBIN`
The parameter `MAX_MAP_SIZE` specifies the maximum number of sets of users or groups for which the function can maintain `ROUND_ROBIN` counts. The dynamic assignment function holds a list of users and groups in memory for each group (or list of users and groups) on which it is asked to execute the `ROUND_ROBIN` function.
- `MOST_PRODUCTIVE`
The parameter `DEAFULT_TIME_PERIOD` specified the length of time (in days) over which to calculate the user's productivity. This value can be overridden when calling the `MOST_PRODUCTIVE` dynamic assignment function. Use an XPath function by specifying an alternative value as the third parameter in the XPath function call.

For more information about configuring the dynamic assignment functions from Oracle Enterprise Manager Fusion Middleware Control Console, see *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite*.

29.3.1.3 How to Configure Display Names for Dynamic Assignment Functions

The runtime config service provides methods for returning a list of available user and group dynamic assignment functions. These functions return both the name of the function, and a user-displayable label for the function. The functions support localization of the display name, so that it displays in the appropriate language for the context user. These functions are used by Oracle BPM Worklist to show a list of available dynamic assignment functions. This applies exclusively to dynamic assignment functions. Display names for task escalation functions are not supported.

To configure display names for dynamic assignment functions:

1. Specify display names (and appropriate translations) for your dynamic assignment functions by adding entries to the resource property file `WorkflowLabels.properties`, and associated resource property files in other languages. This file exists in the classpath identified in the workflow configuration parameter `workflowCustomizationsClasspathURL`.

Entries for dynamic assignment functions must be of the form:

```
DYN_ASSIGN_FN.[function name]=Function Display Name
```

For instance, the entry for the `ROUND_ROBIN` function is:

```
DYN_ASSIGN_FN.ROUND_ROBIN = Round Robin
```

Note that adding entries to these files for dynamic assignment functions is optional. If no entry is present in the file, then the name of the function (for example, `ROUND_ROBIN`) is used instead.

For more information about the `WorkflowLabels.properties` file, see the `workflow-110-workflowCustomizations` sample available at the following URL:

http://www.oracle.com/technology/sample_code/products/hwf

29.3.1.4 How to Implement a Task Escalation Function

Task escalation functions are very similar to dynamic assignment functions, but perform a different function (determining to whom a task is assigned when it is escalated). Custom implementations must implement a different interface (`IDynamicTaskEscaltionFunction`).

29.3.2 Dynamically Assigning Task Participants with the Assignment Service

Human workflow participants are specified declaratively in a routing slip. The routing slip guides the human workflow by specifying the participants and how they participate in the human workflow (for example, management chain hierarchy, sequential list of approvers, and so on).

The Human Task Editor enables you to declaratively create the routing slip using various built-in patterns. In addition, you can use advanced routing based on business rules to do more complex routing. However, if you want to do more sophisticated routing using custom logic, then you implement a custom assignment service to do routing. To support a dynamic assignment, an assignment service is used. The assignment service is responsible for determining the task assignees. You can also

implement your own assignment service and plug in that implementation for use with a particular human workflow.

The assignment service determines the following task assignment details in a human workflow:

- The assignment when the task is initiated.
- The assignment when the task is reinitiated.
- The assignment when a user updates the task outcome. When the task outcome is updated, the task can either be routed to other users or completed.
- The assignees from whom information for the task can be requested.
- If the task supports reapproval from Oracle BPM Worklist, a user can request information for reapproval.
- The users who reapprove the task if reapproval is supported.

The human workflow service identifies and invokes the assignment service for a particular task to determine the task assignment.

For example, a simple assignment service iteration is as follows:

1. A client initiates an expense approval task whose routing is determined by the assignment service.
2. The assignment service determines that the task assignee is `jcooper`.
3. When `jcooper` approves the task, the assignment service assigns the task to `jstein`. The assignment service also specifies that a notification must be sent to the creator of the task, `jlondon`.
4. `jstein` approves the task and the assignment service indicates that there are no more users to which to assign the task.

29.3.2.1 How to Implement an Assignment Service

To implement an assignment service:

1. Implement the assignment service with the `IAssignmentService` interface. The human workflow service passes the following information to the assignment service to determine the task assignment:
 - Task document
The task document that is executed by the human workflow. The task document contains the payload and other task information like current state, and so on.
 - Map of properties
When an assignment service is specified, a list of properties can also be specified to correlate callbacks with back-end services that determine the task assignees.
 - Task history
The task history is a list of chronologically-ordered task documents to trace the history of the task. The task documents in this list contain a subset of attributes in the actual task (such as `state`, `updatedBy`, `outcome`, `updatedAt`, and so on).

29.3.2.2 Example of Assignment Service Implementation

Notes:

- The assignment service class cannot be stateful because every time human workflow services must call the assignment service, it creates a new instance.
 - The `getAssigneesToRequestForInformation` method can be called multiple times because one of the criteria to show the request-for-information action is that there are users to request information. Therefore, this method is called every time the human workflow service tries to determine the permitted actions for a task.
-
-

You can implement your own assignment service plug-in that the human workflow service invokes during human workflow execution.

[Example 29-4](#) provides a sample `IAssignmentService` implementation named `TestAssignmentService.java`.

Example 29-4 Sample IAssignmentService Implementation

```

/* $Header: TestAssignmentService.java 24-may-2006.18:26:16 Exp $ */
/* Copyright (c) 2004, 2006, Oracle. All rights reserved. */
/*
DESCRIPTION
Interface IAssignmentService defines the callbacks an assignment
service will implement. The implementation of the IAssignmentService
will be called by the workflow service
PRIVATE CLASSES
<list of private classes defined - with one-line descriptions>
NOTES
<other useful comments, qualifications, etc.>
MODIFIED      (MM/DD/YY)

*/
/**
 * @version $Header: IAssignmentService.java 29-jun-2004.21:10:35 Exp
 * $
 *
 *
 */
package oracle.bpel.services.workflow.test.workflow;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import oracle.bpel.services.workflow.metadata.routing slip.model.*;
import oracle.bpel.services.workflow.metadata.routing slip.model.Participants;
import
oracle.bpel.services.workflow.metadata.routing slip.model.ParticipantsType.*;
import oracle.bpel.services.workflow.task.IAssignmentService;
import oracle.bpel.services.workflow.task.ITaskAssignee;
import oracle.bpel.services.workflow.task.model.Task;
public class TestAssignmentService implements
oracle.bpel.services.workflow.task.IAssignmentService {
    static int numberOfApprovals = 0;
    static String[] users = new String[]{"jstein", "wfaulk", "cdickens"};
    public Participants onInitiation(Task task,

```

```

        Map propertyBag) {
            return createParticipant();
        }
    public Participants onReinitiation(Task task,
        Map propertyBag) {
        return null;
    }
    public Participants onOutcomeUpdated(Task task,
        Map propertyBag,
        String updatedBy,
        String outcome) {
        return createParticipant();
    }
    public Participants onAssignmentSkipped(Task task,
        Map propertyBag) {
        return null;
    }
    public List getAssigneesToRequestForInformation(Task task,
        Map propertyBag) {
        List rfiUsers = new ArrayList();
        rfiUsers.add("jcooper");
        rfiUsers.add("jstein");
        rfiUsers.add("wfaulk");
        rfiUsers.add("cdickens");
        return rfiUsers;
    }
    public List getReapprovalAssignees(Task task,
        Map propertyBag,
        ITaskAssignee infoRequestedAssignee) {
        List reapprovalUsers = new ArrayList();
        reapprovalUsers.add("jstein");
        reapprovalUsers.add("wfaulk");
        reapprovalUsers.add("cdickens");
        return reapprovalUsers;
    }
    private Participants createParticipant() {
        if (numberOfApprovals > 2) {
            numberOfApprovals = 0;
            return null;
        }
        String user = users[numberOfApprovals++];

        ObjectFactory objFactory = new ObjectFactory();
        Participants participants = objFactory.createParticipants();
        Participant participant = objFactory.createParticipantsTypeParticipant();
        participant.setName("Loan Agent");
        ResourceType resource2 = objFactory.createResourceType(user);
        resource2.setIsGroup(false);
        resource2.setType("STATIC");
        participant.getResource().add(resource2);

        participants.getParticipantOrSequentialParticipantOrAdhoc().
            add(participant);
        return participants;
    }
}

```

29.3.2.3 How to Deploy a Custom Assignment Service

To deploy a custom assignment service:

1. Use one of the following methods to make an assignment service implementation class and its related classes available in the class path of Oracle BPEL Process Manager:
 - Load your classes in `SCA-INF/classes` directly or `SCA-INF/lib` as a JAR.
 - Change the Oracle BPEL Process Manager shared library to include your JAR files.

Note:

- You cannot create different versions of the assignment service for use in different BPEL processes unless you change package names or class names.
 - Java classes and JAR files in the suitcase are not available in the class path and therefore cannot be used as a deployment model for the assignment service.
 - The steps must be repeated for each node in a cluster.
-
-

29.3.3 Custom Escalation Function

The custom escalation function enables you to integrate a custom rule in a human workflow.

To implement a custom escalation function:

1. Create a custom task escalation function and register this with the human workflow service that uses that function in task definitions.
2. Use the **Advanced Settings** section of the Human Task Editor to integrate the rule in a human workflow.

For more information, see [Section 25.3.10.1, "Specifying Escalation Rules."](#)

29.4 Class Loading for Callbacks and Resource Bundles

You can load classes for the following callbacks and resource bundles directly from the SOA project instead of having to load classes in the `oracle.soainfra.common` shared library and restarting Oracle WebLogic Server:

- `IAssignmentService`
- `IRestrictedAssignmentService`
- `IRoutingSlipCallback`
- `IPercentageCompletionCallback`
- `INotificationCallback`
- Project level resource bundles

The callback classes can be in the following locations:

- JARs in the `SCA-INF/lib` directory of the project
- Classes in the `SCA-INF/classes` directory of the project

Additionally, to support backward compatibility, the project level resource bundles can also be in the same directory as the `.task` file.

29.5 Resource Bundles in Workflow Services

This section describes the resource bundles used in human workflow services and how they can be customized to provide alternative resource strings.

The human workflow service APIs and Oracle BPM Worklist use the locale set in the `IWorkflowContext` object to access the APIs. This is the locale of the user in the user directory configured with the identity service. If no locale is specified for the user, then the default locale for the Java EE server is used instead.

It is possible for API clients to override this locale by setting a new value in the `IWorkflowContext` object. Oracle BPM Worklist provides a user preference option that allows users to use their browser's locale, rather than the locale set in their user directory.

29.5.1 Task Resource Bundles

Each human workflow component can be associated with a resource bundle. The bundle defines the resource strings to be used as display names for the task outcomes. The resource strings are returned by the `TaskMetadataService` method `getTaskDefinitionOutcomes`, and are displayed in Oracle BPM Worklist and the task flow task details application.

In addition, you can use the human workflow XPath function `getTaskResourceBundle` string to look up resource strings for the task's resource bundle. For example, this XPath function can be used as part of the XPath expression used to construct notification messages for the task.

A human workflow component is associated with a resource bundle by setting the **Resource Name** and **Resource Location** fields of the Resource Details dialog in the **Advanced Settings** section of the Human Task Editor in Oracle JDeveloper. Note that the value for the **Resource Location** field is a URL, and the resource bundle can be contained within a JAR file pointed to by the URL. It is possible to share the same resource bundle between multiple human workflow components by using a common location for the resource bundle.

If no resource bundle is specified for the human workflow component, the resource string is looked up in the global resource bundle. (See [Section 29.5.2, "Global Resource Bundle – WorkflowLabels.properties."](#)) Commonly-used task outcomes can be defined in the global resource bundle, alleviating the need to define a resource bundle for individual human workflow components.

If no resource string can be located for a particular outcome, then the outcome name is used as the display value in all locales.

29.5.2 Global Resource Bundle – WorkflowLabels.properties

The following global resource bundle is used by human workflow service APIs to look up resource strings:

```
oracle.bpel.services.workflow.resource.WorkflowLabels.properties
```

You can customize this bundle to provide alternatives for existing display strings or to add additional strings (for example, for flex field attribute labels, standard views, or custom dynamic assignment functions).

The global resource bundle provides resource strings for the following:

- Task attributes:

Labels for the various task attributes displayed in Oracle BPM Worklist (or other clients). Resource string values are returned from the following `TaskMetadataService` methods:

- `getTaskAttributes`
- `getTaskAttributesForTaskDefinition`
- `getTaskAttributesForTaskDefinitions`

- Flex field attribute labels:

Labels for flex field attribute labels created through the runtime config service. These strings are used in Oracle BPM Worklist when displaying mapped flex field attributes. Resource string values are returned from the `TaskMetadataService` methods:

- `getTaskAttributesForTaskDefinition`
- `getTaskAttributesForTaskDefinitions`

If translated resource strings are required for flex field mappings, then customize the `WorkflowLabels.properties` bundle to include the appropriate strings.

- Task outcomes:

Default resource strings for common task outcomes. These can be overridden by the task resource bundle, as described above. The resource strings are returned by the `TaskMetadataService` method `getTaskDefinitionOutcomes`, if no task-specific resource bundle has been specified. As shipped, the global resource bundle contains resource strings for the following outcomes:

- Approve
- Reject
- Yes
- No
- OK
- Defer
- Accept
- Acknowledge

- Dynamic assignment function names:

Labels for dynamic assignment functions. These strings are returned from the runtime config service methods `getUserDynamicAssignmentFunctions` and `getGroupDynamicAssignmentFunctions`. The shipped resource bundle contains labels for the standard dynamic assignment functions (`ROUND_ROBIN`, `LEAST_BUSY`, and `MOST_PRODUCTIVE`). If additional custom dynamic assignment functions have been created, then modify the `WorkflowLabels.properties` resource bundle to provide resource strings for the new functions.

- Standard view names:

Labels for standard views. If you want translated resource strings for any standard views you create, then add them here. Standard view resource strings are looked

up from the resource bundle, and are returned as the standard view name from the `UserMetadataService` methods `getStandardTaskViewList` and `getStandardTaskViewDetails`. The key for the resource string should be the name given to the standard view when it is created. If no resource string is added for a particular standard view, then the name as entered is used instead.

- Notification messages:

Resource strings used when the task service sends automatic notifications. These can be customized to suit user requirements.

- Task routing error comments:

When an error is encountered in the routing of a task, the task service automatically appends comments to the task to describe the error. The various strings used for the comments are defined in this resource bundle.

A copy of the `WorkflowLabels.properties` resource bundle is available in the sample `workflow-110-workflowCustomizations`.

You can customize the `WorkflowLabels.properties` resource bundle by editing it and then adding the customized version to the class path for workflow services, ahead of the version that ships with the product.

This can be done in the following ways:

- Place the customized file in a directory tree:

`directory_path/oracle/bpel/services/workflow/resource/WorkflowLabels.properties`

- Update the `worklflowCustomClasspathURL` configuration parameter to point to `directory_path` (As this is a URL, it is possible to host the resource bundles on a web server, and make them accessible to multiple Oracle WebLogic Servers). This approach is described in detail in sample `workflow-110-workflowCustomizations`. To download this sample, visit the following URL:

http://www.oracle.com/technology/sample_code/products/hwf

29.5.3 Worklist Client Resource Bundles

The ADF worklist client application uses two resource bundles that contain all the strings displayed in the worklist client web application.

- `oracle.bpel.worklistapp.resource.WorkflowResourceBundle`: This contains strings used by both the ADF Oracle BPM Worklist, and the JSP-based sample Oracle BPM Worklist that shipped with version 10.1.3 of Oracle SOA Suite.
- `oracle.bpel.worklistapp.resource.WorklistResourceBundle`. This contains strings used only by the ADF Oracle BPM Worklist.

Copies of the worklist resource bundles are available in the sample `workflow-110-workflowCustomizations`.

The sample illustrates how to customize Oracle BPM Worklist by recompiling these resource bundles, and adding the updated classes to Oracle BPM Worklist.

29.5.4 Task Detail ADF Task Flow Resource Bundles

The ADF task flow applications and associated data controls that get created to display the details of a particular task type use the resource bundle `oracle.bpel.services.workflow.worklist.resource.worklist` to store their resource strings.

You can provide your own custom resource strings for a task detail ADF task flow by adding a customized resource bundle in the task flow application.

A copy of the `WorkflowLabels.properties` resource bundle is available in the sample `workflow-110-workflowCustomizations`. This sample illustrates in detail how to provide your own customized resource strings for the task detail ADF task flow application.

29.5.5 Case Sensitivity

By default, the human workflow system is case insensitive to user names. All user names are stored in lowercase. However, group names and application role names are always case sensitive. User name case insensitivity can be changed in Oracle Enterprise Manager Fusion Middleware Control Console.

Caution: Only change this setting after performing a new installation. Changing this value on an installation that is actively processing instances, or has many instances in the database, causes serious issues.

To change case sensitivity:

1. Log in to Oracle Enterprise Manager Fusion Middleware Control Console.
2. In the navigator, expand the **SOA** folder.
3. Right-click **soa-infra**, and select **Administration > System Mbean Browser**.
The System MBean Browser displays on the right side of the page.
4. Expand **Application Defined MBeans > oracle.as.soainfra.config > Server: *server_name* > WorkflowIdentityConfig > human-workflow > WorkflowIdentityConfig.PropertyType**.
5. Click **caseSensitive**.
6. Click the **Operations** tab.
7. Click **setValue**.
8. In the **Value** field, enter `true`, and click **Invoke**.

If you are upgrading from 10.1.3, which by default was case sensitive, set **caseSensitive** to `true` for the system to be the same as with 10.1.3.

29.6 Introduction to Human Workflow Client Integration with Oracle WebLogic Server Services

This section describes how human workflow clients integrate with Oracle WebLogic Server services.

29.6.1 Human Workflow Services Clients

Human workflow services expose the following workflow services:

- Task service
- Task query service
- User metadata service

- Task evidence service
- Task metadata service
- Runtime config service
- Task report service

To use any of these services, you must use the abstract factory pattern for workflow services. The abstract factory pattern provides a way to encapsulate a group of individual factories that have a common theme.

Perform the following tasks:

- Get the `IWorkflowServiceClient` instance for the specific service type. The `WorkflowServiceClientFactory` provides a static factory method to get `IWorkflowServiceClient` according to the service type.
- Use the `IWorkflowServiceClient` instance to get the service instance to use.

There are three supported service types:

- Local
- Remote
- SOAP

Local and remote clients use Enterprise JavaBeans clients (local Enterprise JavaBeans and remote Enterprise JavaBeans, accordingly). SOAP uses SOAP clients. Each type of service requires you to configure workflow clients. [Example 29–5](#) provides details.

Example 29–5 Client Configuration File

```
<workflowServicesClientConfiguration>
<server name="default" default="true">
  <localClient>
    <participateInClientTransaction>>false</participateInClientTransaction>
  </localClient>
  <remoteClient>
    <serverURL>t3://myhost.us.oracle.com:7001</serverURL>
    <userName>weblogic</userName>
    <password>weblogic</password>
    <initialContextFactory>weblogic.jndi.WLInitialContextFactory
      </initialContextFactory>
    <participateInClientTransaction>>false</participateInClientTransaction>
  </remoteClient>
  <soapClient>
    <rootEndPointURL>http://myhost.us.oracle.com:7001</rootEndPointURL>
    <identityPropagation mode="dynamic" type="saml">
      <policy-references>
        <policy-reference enabled="true" category="security"
          uri="oracle/wss10_saml_token_client_policy"/>
      </policy-references>
    </identityPropagation>
  </soapClient>
</server>
</workflowServicesClientConfiguration>
```

29.6.1.1 Task Query Service Client Code

[Example 29–6](#) provides an example of the task query service client code.

Example 29–6 Task Query Service Client Code

```

/**
 * WFClientSample
 */
package oracle.bpel.services.workflow.samples;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import oracle.bpel.services.workflow.IWorkflowConstants;
import oracle.bpel.services.workflow.WorkflowException;
import oracle.bpel.services.workflow.client.IWorkflowServiceClient;
import oracle.bpel.services.workflow.client.WorkflowServiceClientFactory;
import oracle.bpel.services.workflow.client.IWorkflowServiceClientConstants
    .CONNECTION_PROPERTY;
import oracle.bpel.services.workflow.query.ITaskQueryService;
import oracle.bpel.services.workflow.query.ITaskQueryService.AssignmentFilter;
import oracle.bpel.services.workflow.query.ITaskQueryService.OptionalInfo;
import oracle.bpel.services.workflow.repos.Ordering;
import oracle.bpel.services.workflow.repos.Predicate;
import oracle.bpel.services.workflow.repos.TableConstants;
import oracle.bpel.services.workflow.verification.IWorkflowContext;

public class WFClientSample {

    public static List runClient(String clientType) throws WorkflowException {
        try {

            IWorkflowServiceClient wfSvcClient = null;
            ITaskQueryService taskQuerySvc = null;
            IWorkflowContext wfCtx = null;

            // 1. this step is optional since configuration can be set in wf_client_
            //    config.xml file
            Map<CONNECTION_PROPERTY, String> properties = new HashMap<CONNECTION_
PROPERTY, String>();
            if (WorkflowServiceClientFactory.REMOTE_CLIENT.equals(clientType)) {
                properties.put(CONNECTION_PROPERTY.EJB_INITIAL_CONTEXT_FACTORY,
"weblogic.jndi.WLInitialContextFactory");
                properties.put(CONNECTION_PROPERTY.EJB_PROVIDER_URL,
"t3://myhost.us.oracle.com:7001");
                properties.put(CONNECTION_PROPERTY.EJB_SECURITY_CREDENTIALS,
"weblogic");
                properties.put(CONNECTION_PROPERTY.EJB_SECURITY_PRINCIPAL, "weblogic");
            } else if (WorkflowServiceClientFactory.SOAP_CLIENT.equals(clientType)) {
                properties.put(CONNECTION_PROPERTY.SOAP_END_POINT_ROOT,
"http://myhost:7001");
                properties.put(CONNECTION_PROPERTY.SOAP_IDENTITY_
PROPAGATION, "non-saml"); // optional
            }
            // 2. gets IWorkflowServiceClient for specified client type
            wfSvcClient =
WorkflowServiceClientFactory.getWorkflowServiceClient(clientType, properties,
null);

            // 3. gets ITaskQueryService instance
            taskQuerySvc = wfSvcClient.getTaskQueryService();

```

```

// 4. gets IWorkflowContext instance
wfCtx = taskQuerySvc.authenticate("jcooper", "welcome1".toCharArray(),
"jazn.com");

// 5. creates displayColumns
List<String> displayColumns = new ArrayList<String>(8);
displayColumns.add("TASKID");
displayColumns.add("TASKNUMBER");
displayColumns.add("TITLE");
displayColumns.add("CATEGORY");

// 6. creates optionalInfo
List<ITaskQueryService.OptionalInfo> optionalInfo = new
ArrayList<ITaskQueryService.OptionalInfo>();
optionalInfo.add(ITaskQueryService.OptionalInfo.DISPLAY_INFO);

// 7. creates assignmentFilter
AssignmentFilter assignmentFilter = AssignmentFilter.MY_AND_GROUP;

// 8. creates predicate
List<String> stateList = new ArrayList<String>();
stateList.add(IWorkflowConstants.TASK_STATE_ASSIGNED);
stateList.add(IWorkflowConstants.TASK_STATE_INFO_REQUESTED);
Predicate predicate = new Predicate(TableConstants.WFTASK_STATE_COLUMN,
Predicate.OP_IN, stateList);

// 9. creates ordering
Ordering ordering = new Ordering(TableConstants.WFTASK_DUEDATE_COLUMN,
true, false);
ordering.addClause(TableConstants.WFTASK_CREATEDDATE_COLUMN, true,
false);

// 10. calls service - query tasks
List taskList = taskQuerySvc.queryTasks(wfCtx,
(List<String>) displayColumns,
(List<OptionalInfo>) optionalInfo,
(AssignmentFilter)
assignmentFilter,
(String) null, // keywords is
optional (see javadoc)
// optional
predicate,
ordering,
0, // starting row
100); // ending row for paging, 0
if no paging

// Enjoy result
System.out.println("Successfully get list of tasks for client type: " +
clientType +
". The list size is " + taskList.size());
return taskList;
} catch (WorkflowException e) {
System.out.println("Error occurred");
e.printStackTrace();
throw e;
}
}

public static void main(String args[]) throws Exception {

```

```

        runClient(WorkflowServiceClientFactory.REMOTE_CLIENT);
        runClient(WorkflowServiceClientFactory.SOAP_CLIENT);
    }
}

```

29.6.1.2 Configuration Option

Each type of client is required to have a workflow client configuration. You can set the configuration in the following locations:

- wf_client_config.xml file
- Property map

The property map is always complementary to the wf_client_config.xml file. The property map can overwrite the configuration attribute. The file is optional. If it cannot be found in the application classpath, then the property map is the main source of configuration.

29.6.1.2.1 Workflow Client Configuration File - wf_client_config.xml The client configuration XSD schema is present in the wf_client_config.xsd file.

The server configuration should contain three types of clients:

- localClient
- remoteClient
- soapClient

Oracle recommends that you specify all clients. This is because some services (for example, the identity service) do not have remote and local clients. Therefore, when you use remote clients for other services, the identity service uses the SOAP service.

An example of a client configuration XML file is shown in [Example 29-7](#). The configuration defines a server named default. The XML file must go into the client application's EAR file.

Example 29-7 Client Configuration

```

<workflowServicesClientConfiguration>
server name="default" default="true">
<localClient>
    <participateInClientTransaction>>false</participateInClientTransaction>
</localClient>

<remoteClient>
    <serverURL>t3://myhost.us.oracle.com:7001</serverURL>
    <userName>weblogic</userName>
    <password>weblogic</password>
    <initialContextFactory>weblogic.jndi.WLInitialContextFactory
    </initialContextFactory>
    <participateInClientTransaction>>false</participateInClientTransaction>
</remoteClient>

<soapClient>
    <rootEndPointURL>http://myhost.us.oracle.com:7001</rootEndPointURL>
    <identityPropagation mode="dynamic" type="saml">
<policy-references>
    <policy-reference enabled="true" category="security"
        uri="oracle/wss10_saml_token_client_policy"/>
</policy-references>

```

```

    </identityPropagatation>
</soapClient>

</server>
</workflowServicesClientConfiguration>

```

29.6.1.2.2 Workflow Client Configuration in the Property Map If you want to specify the connection property dynamically, you can use a `java.util.Map` to specify the properties. The properties take precedence over definitions in the configuration file. Therefore, the values of the properties overwrite the values defined in `wf_client_config.xml`. If you do not want to dynamically specify connection details to the server, you can omit the property setting in the map and pass a null value to the factory method. In that case, the configuration `wf_client_config.xml` is searched for in the client application class path.

The configuration file must be in the class path only if you want to get the configuration from the file. It is optional to have the file if all settings from the specific client type are done through the property map. The JAXB object is also not required to have the file, since all settings are taken from the JAXB object.

```

IWorkflowServiceClient wfSvcClient =
WorkflowServiceClientFactory.getWorkflowServiceClient(WorkflowServiceClientFactory
.REMOTE_CLIENT,
(Map<IWorkflowServiceClientConstants.CONNECTION_PROPERTY, String> ) null, null);

```

If you do so, the value from `wf_client_config.xml` found in the class path is used by the client to access the services. If the file is not found in the class path and you do not provide the setting according to the service type, a workflow exception is thrown. If the properties map is null and the file is not found, an exception is thrown. If the client omits some properties in the map while the file is not found, the service call fails at runtime (the properties are complementary to the file).

The `IWorkflowServiceClientConstants.CONNECTION_PROPERTY`, which can be used in the properties map, is show in [Example 29-8](#):

Example 29-8 CONNECTION_PROPERTY

```

public enum CONNECTION_PROPERTY {
    MODE, // not supported , deprecated
    EJB_INITIAL_CONTEXT_FACTORY,
    EJB_PROVIDER_URL,
    EJB_SECURITY_PRINCIPAL,
    EJB_SECURITY_CREDENTIALS,
    // SOAP configuration
    SOAP_END_POINT_ROOT,
    SOAP_IDENTITY_PROPAGATION, // if value is 'saml' then SAML-token
    identity propagatation is used
    SOAP_IDENTITY_PROPAGATION_MODE, // "dynamic"
    MANAGEMENT_POLICY_URI, // dafault value is "oracle/log_policy"
    SECURITY_POLICY_URI, // default value is "oracle/wss10_
    saml_token_client_policy"
    // LOCAL and REMOTE EJB option
    TASK_SERVICE_PARTICIPATE_IN_CLIENT_TRANSACTION // default value is
    false
    //(task service EJB will start a new transaction)
};

```

Note: If you use the properties map, you do not need to specify `IWorkflowServiceClientConstants.CONNECTION_PROPERTY.MODE`. This property has been deprecated in this release.

[Example 29–9](#) provides an example for remote Enterprise JavaBeans clients.

Example 29–9 Example for Remote Enterprise JavaBeans Clients

```
Map<CONNECTION_PROPERTY,String> properties = new HashMap<CONNECTION_
PROPERTY,String>();
properties.put(CONNECTION_PROPERTY.EJB_INITIAL_CONTEXT_
FACTORY,"weblogic.jndi.WLInitialContextFactory");

properties.put(CONNECTION_PROPERTY.EJB_PROVIDER_URL,
    "t3://myhost.us.oracle.com:7001");
properties.put(CONNECTION_PROPERTY.EJB_SECURITY_PRINCIPAL, "weblogic");
properties.put(CONNECTION_PROPERTY.EJB_SECURITY_CREDENTIALS, "weblogic");
IWorkflowServiceClient client =
    WorkflowServiceClientFactory.getWorkflowServiceClient(
        WorkflowServiceClientFactory.REMOTE_CLIENT,
        properties, null);
```

[Example 29–10](#) provides an example for a SOAP client.

Example 29–10 Example for SOAP Client

```
Map<CONNECTION_PROPERTY,String> properties = new HashMap<CONNECTION_
PROPERTY,String>();
properties.put(CONNECTION_PROPERTY.SOAP_END_POINT_ROOT, "http://myhost:7001");
IWorkflowServiceClient client =
    WorkflowServiceClientFactory.getWorkflowServiceClient(
        WorkflowServiceClientFactory.SOAP_CLIENT,
        properties, null);
```

29.6.1.3 Client Logging

Clients can optionally pass in a `java.util.logging.Logger` to where the client logs messages. If there is no logger specified, the workflow service client code does not log anything. [Example 29–11](#) shows how to pass a logger to the workflow service clients:

Example 29–11 `java.util.logging.Logger`

```
java.util.logging.Logger logger = ....;

IWorkflowServiceClient client =
    WorkflowServiceClientFactory.getWorkflowServiceClient(WorkflowServiceClientFactory
.REMOTE_CLIENT, properties, logger);
```

29.6.1.4 Configuration Migration Utility

The client configuration schema has changed between release 10.1.3.x and 11g Release 1. To migrate from release 10.1.3.x to 11g Release 1, use the following utility:

```
java -classpath wsclient_extended.jar:bpm-services.jar
    oracle.bpel.services.workflow.client.config.MigrateClientConfiguration
    original_file [new_file];
```

where *original_file* is a `wf_client_config.xml` file from 10.1.3.x and *new_file* is the optional name of the new configuration file. If a new name is not specified, the utility backs up the original configuration file and overwrites the `wf_client_config.xml` file.

29.6.2 Identity Propagation

This section describes how to propagate identities using Enterprise JavaBeans and SAML-tokens for SOAP clients.

There are performance implications for getting the workflow context for every request. This is also true for identity propagation. If you use identity propagation with SAML-token or Enterprise JavaBeans, authenticate the client by passing null for the user and password, get the workflow context instance, and use another service call with workflow context without identity propagation.

29.6.2.1 Enterprise JavaBeans Identity Propagation

The client application can propagate user identity to services by using Enterprise JavaBeans identity propagation. The client code is responsible for securing the user identity.

29.6.2.1.1 Client Configuration If you use identity propagation, the client code must omit the element's `<userName>` and `<password>` under the `<remoteClient>` element in the `wf_client_config.xml` configuration file. In addition, do not populate the following properties into

`Map<IWorkflowServiceClientConstants.CONNECTION_PROPERTY, String>` properties as you did in [Section 29.6.1.2.2, "Workflow Client Configuration in the Property Map."](#)

- `IWorkflowServiceClientConstants.CONNECTION_PROPERTY.EJB_SECURITY_PRINCIPAL`
- `IWorkflowServiceClientConstants.CONNECTION_PROPERTY.EJB_SECURITY_CREDENTIALS`

29.6.2.1.2 Requirements for Client Applications For Identity Propagation Identity propagation only works if the application is deployed under the Oracle WebLogic Server container and secured with container security or the client is secured with a custom JAAS login module.

End users log in to the client application with the correct user name and password. The users using the client application must be available in the identity store used by the SOA application. As a best practice, configure the client to use the same identity store as the workflow services and Oracle SOA Suite are using. This guarantees that if the user exists on the client side, they also exist on the server side.

For information about configuring the identity store, see *Oracle Fusion Middleware Security Guide*.

29.6.2.2 SAML Token Identity Propagation for SOAP Client

If you use a SOAP client, you can use the SAML-token identity propagation supported by Oracle web services.

This section assumes the application resides in and is secured by the Oracle WebLogic Server container.

29.6.2.2.1 Client configuration To enable identity propagation, the client configuration must specify a special propagation mode.

29.6.2.2.2 Identity Propagation Mode Setting Through Properties If properties are used, then populate the property `CONNECTION_PROPERTY.SOAP_IDENTITY_PROPAGATION` with the value `saml`.

- Dynamic SAML token propagation mode

The SAML token policy is provided dynamically (the default). The following property is optional. If the identity propagation mode is set, you run by default in dynamic mode.

```
properties.put(IWorkflowServiceClientConstants.CONNECTION_PROPERTY.SOAP_IDENTITY_PROPAGATION_MODE , "dynamic");
```

By default, SAML-token constructs dynamic policy based on the following security policy URI: `oracle/wss10_saml_token_client_policy`. Logging is not used. To overwrite the default policy URI, the client can add the following code:

```
properties.put(CONNECTION_PROPERTY.SECURITY_POLICY_URI      "oracle/wss10_saml_token_client_policy");
properties.put(CONNECTION_PROPERTY.MANAGEMENT_POLICY_URI , "oracle/log_policy");
```

[Example 29–12](#) shows the SAML token dynamic client.

Example 29–12 Token Dynamic Client

```
Map<CONNECTION_PROPERTY,String> properties = new HashMap<CONNECTION_PROPERTY,String>();
properties.put(CONNECTION_PROPERTY.SOAP_IDENTITY_PROPAGATION , "saml");
properties.put(CONNECTION_PROPERTY.SOAP_END_POINT_ROOT,
    "http://myhost.us.oracle.com:7001");
properties.put(CONNECTION_PROPERTY.SECURITY_POLICY_URI, "oracle/wss10_saml_token_client_policy"); //optional
properties.put(CONNECTION_PROPERTY.MANAGEMENT_POLICY_URI , "oracle/log_policy");
//optional
IWorkflowServiceClient client =
    WorkflowServiceClientFactory.getWorkflowServiceClient(
        WorkflowServiceClientFactory.SOAP_CLIENT,
properties, null);
```

The client reference to the policy URI must match the server policy URI. Otherwise, SAML token propagation fails.

29.6.2.2.3 Identity Propagation Mode Setting in Configuration File In the configuration file, you can define the propagation mode by using the `<identityPropagation>` element in the `<soapClient>`, as shown in [Example 29–13](#).

Example 29–13 <identityPropagation> Element

```
<soapClient>
  <rootEndPointURL>http://myhost.us.oracle.com:7001</rootEndPointURL>
  <identityPropagation mode="dynamic" type="saml">
    <policy-references>
      <policy-reference enabled="true" category="security"
uri="oracle/wss10_saml_token_client_policy"/>
    </policy-references>
  </identityPropagation>
</soapClient>
```

For more information, see *Oracle Fusion Middleware Security and Administrator's Guide for Web Services*.

29.6.2.2.4 Identity Propagation Mode Setting Through the JAXB Object You can programmatically set the identity propagation mode with the JAXB object.

29.6.3 Client JAR Files

A client application without identity propagation must have the `bpm-services.jar` file in its class path. For 11g Release 1, the client classpath requires the following files:

```
{bea.home}/wlserver_10.3/server/lib/wlfullclient.jar
{bea.home}/AS11gR1SOA/webservices/wsclient_extended.jar
{bea.home}/AS11gR1SOA/soa/modules/oracle.soa.fabric_11.1.1/bpm-infra.jar
{bea.home}/AS11gR1SOA/soa/modules/oracle.soa.workflow_11.1.1/bpm-services.jar
```

The `wlfullclient.jar` file must be generated.

1. Generate the `wlfullclient.jar` as follows:

```
cd {bea.home}/wlserver_10.3/server/lib
java -jar ../../../../modules/com.bea.core.jarbuilder_1.3.0.0.jar
```

29.7 Database Views for Oracle Workflow

This section describes database views that enable queries against the Oracle workflow services schema to receive reports. [Table 29–22](#) lists the reports exposed in Oracle BPM Worklist and the database views corresponding to these reports.

Table 29–22 Report Views

Existing Worklist Report	Corresponding Database View
Unattended Tasks report	WFUNATTENDEDTASKS_VIEW
Task Cycle Time report	WFTASKCYCLETIME_VIEW
Task Productivity report	WFPRODUCTIVITY_VIEW
Task Priority Report	WFTASKPRIORITY_VIEW

29.7.1 Unattended Tasks Report View

[Table 29–23](#) describes the `WFUNATTENDEDTASKS_VIEW` report view.

Table 29–23 Unattended Tasks Report View

Name	Type
TASKID ¹	VARCHAR2 (64)
TASKNAME	VARCHAR2 (200)
TASKNUMBER	NUMBER
CREATEDDATE	DATE
EXPIRATIONDATE	DATE
STATE	VARCHAR2 (100)
PRIORITY	NUMBER
ASSIGNEEGROUPS	VARCHAR2 (2000)

¹ NOT NULL column

For example:

- Query unattended tasks that have an expiration date of next week:


```
SELECT tasknumber, taskname, assigneegroups FROM WFUNATTENDEDTASKS_VIEW
WHERE expirationdate > current_date AND expirationdate < current_date +
7;
```
- Query unattended tasks for mygroup:


```
SELECT tasknumber, taskname, assigneegroups FROM WFUNATTENDEDTASKS_VIEW
WHERE 'mygroup' IN assigneegroups;
```
- Query unattended tasks created in the last 30 days:


```
SELECT tasknumber, taskname, assigneegroups FROM WFUNATTENDEDTASKS_VIEW
WHERE createddate > current_date -30;
```

29.7.2 Task Cycle Time Report View

Table 29–24 describes the WFTASKCYCLETIME_VIEW report view.

Table 29–24 Task Cycle Time Report View

Name	Type
TASKID ¹	VARCHAR2 (64)
TASKNAME	VARCHAR2 (200)
TASKNUMBER	NUMBER
CREATEDDATE	DATE
ENDDATE	DATE
CYCLETIME	NUMBER (38)

¹ NOT NULL column

For example:

- Compute the average cycle time (task completion time) for completed tasks that were created in the last 30 days:


```
SELECT avg(cycletime) FROM WFTASKCYCLETIME_VIEW WHERE createddate >
(current_date - 30);
```
- Query the average cycle time for all completed tasks created in the last 30 days and group them by task name:


```
SELECT taskname, avg(cycletime) FROM WFTASKCYCLETIME_VIEW WHERE
createddate > (current_date - 30) GROUP BY taskname;
```
- Query the least and most time taken by each task:


```
SELECT taskname, min(cycletime), max(cycletime) FROM WFTASKCYCLETIME_VIEW
GROUP BY taskname;
```
- Compute the average cycle time for tasks completed in the last seven days:


```
SELECT avg(cycletime) FROM WFTASKCYCLETIME_VIEW WHERE enddate >
(current_date - 7);
```
- Query tasks that took more than seven days to complete:

```
SELECT taskname, avg(cycletime) FROM WFTASKCYCLETIME_VIEW WHERE cycletime
> ((current_date +7) - current_date) GROUP BY taskname;
```

29.7.3 Task Productivity Report View

Table 29–25 describes the WFPRODUCTIVITY_VIEW report view.

Table 29–25 Task Productivity Report View

Name	Type
TASKNAME	VARCHAR2 (200)
TASKID	VARCHAR2 (200)
TASKNUMBER	NUMBER
USERNAME	VARCHAR2 (200)
STATE ¹	VARCHAR2 (100)
LASTUPDATEDDATE	DATE

¹ For completed tasks, the state is null. Use `decode(outcome, '', 'COMPLETED', outcome)` in queries.

For example:

- Count the number of unique tasks that the user has updated in the last 30 days:

```
SELECT username, count(distinct(taskid)) FROM WFPRODUCTIVITY_VIEW WHERE
lastupdateddate > (current_date -30) GROUP BY username;
```

- Count the number of tasks that the user has updated (one task may have been updated multiple times) in the last seven days:

```
SELECT username, count(taskid) FROM WFPRODUCTIVITY_VIEW WHERE
lastupdateddate > (current_date -7) GROUP BY username;
```

- Count the number of tasks of each task type on which the user has worked:

```
SELECT username, taskname, count(taskid) FROM WFPRODUCTIVITY_VIEW GROUP
BY username, taskname;
```

- Count the number of tasks of each task type that the user has worked on in the last 100 days:

```
SELECT username, taskname, count(taskid) FROM WFPRODUCTIVITY_VIEW WHERE
lastupdateddate > (current_date -100) GROUP BY username, taskname;
```

29.7.4 Task Priority Report View

Table 29–26 describes the WFTASKPRIORITY_VIEW report view.

Table 29–26 Task Priority Report View

Name	Type
TASKID ¹	VARCHAR2 (64)
TASKNAME	VARCHAR2 (200)
TASKNUMBER	NUMBER
PRIORITY	NUMBER
OUTCOME	VARCHAR2 (100)

Table 29–26 (Cont.) Task Priority Report View

Name	Type
ASSIGNEDDATE	DATE
UPDATEDDATE	DATE
UPDATEDBY	VARCHAR2 (64)

¹ NOT NULL column

For example:

- Query the number of tasks updated by each user in each task priority:

```
SELECT updatedby, priority, count(taskid) FROM WFTASKPRIORITY_VIEW GROUP
BY updatedby, priority;
```

- Query task-to-outcome distribution:

```
SELECT taskname, decode(outcome, '', 'COMPLETED', outcome), count
(taskid) FROM WFTASKPRIORITY_VIEW GROUP BY taskname, outcome;
```

- Query the number of tasks updated by the given user in each priority:

```
SELECT priority, count(taskid) FROM WFTASKPRIORITY_VIEW WHERE
updatedby='jstein' GROUP BY priority;
```

Integrating Microsoft Excel with a Human Task

Integrating the enterprise system capabilities of Oracle SOA Suite with Microsoft Excel 2007 enables you to:

- Invoke a BPEL process from Microsoft Excel
- Attach Microsoft Excel workbooks to workflow email notifications

You can configure this integration without having to switch between tools.

This chapter contains these sections:

- [Section 30.1, "Configuring Your Environment for Invoking a BPEL Process from an Excel Workbook"](#)
- [Section 30.2, "Attaching Excel Workbooks to Human Task Workflow Email Notifications"](#)

30.1 Configuring Your Environment for Invoking a BPEL Process from an Excel Workbook

From an Excel workbook, you can invoke a BPEL process that is deployed in Oracle WebLogic. To do this, you install a plug-in of the Application Development Framework Desktop Integration (ADF-DI) on the same host as the Excel document that invokes the BPEL process.

To enable this functionality, do the following:

30.1.1 How to Create an JDeveloper Project of the Type Web Service Data Control

You use the Create Web Service Data Control Wizard to create the project.

To create an Oracle JDeveloper project of the type web service data control:

1. In JDeveloper, from the File menu, select **New**. The New Gallery dialog appears.
2. In the Categories section, expand **Business Tier**, then select **Data Controls**. The corresponding items appear in the Items pane.
3. In the Items pane, select **Web Service Data Control** and click **OK**. The Create Web Service Data Control Wizard appears.
4. Follow the instructions in the online Help for this wizard. As you do this, you are prompted to select the WSDL file and operations that will be used for this project.

30.1.2 How to Create a Dummy JSF Page

In this task you generate a page definition file. Note that we are not interested in the actual layout generated in the JSF file, but simply in generating a page definition file which contains these controls and actions. This page definition will be used in Excel file later.

To create a dummy JSF page:

1. In JDeveloper, from the File menu, select **New**. The New Gallery dialog appears.
2. In the Categories section, from the Web Tier node, select JSF. The corresponding items appear in the Items pane.
3. In the Items pane, select JSF Page and then click **OK**. The Create JSF Page dialog appears.
4. Fill in the various fields by following the instructions in the online Help for this dialog.
5. When prompted, drag and drop from the Components Palette the controls and fields you are interested in using in the Excel document.

For an example of how to do this, see "[Task 3: Create a Valid Page Definition File to Be Used in the Excel Workbook](#)" on page 30-12.

30.1.3 How to Add Desktop Integration to Your Oracle JDeveloper Project

To add Oracle ADF Desktop Integration to the technology scope of your project, use the Project Properties dialog box in JDeveloper.

To add Oracle ADF Desktop Integration to your project:

1. In the Application Navigator, right-click the project to which you want to add the Oracle ADF Desktop Integration module and choose **Project Properties** from the context menu.

If your application uses the Fusion Web Application (ADF) application template, you select the **ViewController** project. If your application uses another application template, select the project that corresponds to the web application.

2. In the Project Properties dialog, select **Technology Scope** to view the list of available technologies.
3. Choose the **ADF Desktop Integration** and **ADF Library Web Application Support** project technologies and add them to the Selected Technologies list.
4. Click **OK**.

30.1.4 What Happens When You Add Desktop Integration to Your JDeveloper Project

When you add the Oracle ADF Desktop Integration module to the technology scope of your project, the following events occur:

- The project adds the Oracle ADF Desktop Integration runtime library. This library references the following .jar files in its class path:
 - wsclient.jar
 - adf-desktop-integration.jar
 - resourcebundle.jar
- The project adds an ADF bindings filter (adfBindings).

- The project's deployment descriptor (`web.xml`) is modified to include the following entries:
 - A servlet named `adfdiRemote`

Note: The value for the `url-pattern` attribute of the `servlet-mapping` element for `adfdiRemote` must match the value of the `RemoteServletPath` workbook property described in *Oracle Fusion Middleware Desktop Integration Developer's Guide for Oracle Application Development Framework*.

- A filter named `adfdiExcelDownload`
- A MIME mapping for Excel files (`.xlsx` and `.xlsm`)

The previous list is not exhaustive. Adding Oracle ADF Desktop Integration to a project makes other changes to `web.xml`. Note that some of the entries in `web.xml` will only be added if they do not already exist.

When you add ADF Library Web Application Support to the technology scope of your project, the project's `web.xml` file is modified to include the following entries:

```
<filter>
  <filter-name>ADFLibraryFilter</filter-name>
  <filter-class>oracle.adf.library.webapp.LibraryFilter</filter-class>
</filter>
<filter-mapping>
  <filter-name>ADFLibraryFilter</filter-name>
  <url-pattern>*/*</url-pattern>
  <dispatcher>FORWARD</dispatcher>
  <dispatcher>REQUEST</dispatcher>
</filter-mapping>
<servlet>
  <servlet-name>adflibResources</servlet-name>
  <servlet-class>oracle.adf.library.webapp.ResourceServlet</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>adflibResources</servlet-name>
  <url-pattern>/adflib/*</url-pattern>
</servlet-mapping>
```

Make sure that the filter for ADF Library Web Application Support (`<filter-name>ADFLibraryFilter</filter-name>`) appears below the `adfdiExcelDownload` filter entries in `web.xml` as in the following example so that integrated Excel workbooks can be downloaded from the Fusion web application:

```
<filter>
<filter-name>adfdiExcelDownload</filter-name>
<filter-class>oracle.adf.desktopintegration.filter.DIExcelDownloadFilter</filter-c
lass>
</filter>
<filter>
<filter-name>ADFLibraryFilter</filter-name>
<filter-class>oracle.adf.library.webapp.LibraryFilter</filter-class>
</filter>
...
<filter-mapping>
<filter-name>adfdiExcelDownload</filter-name>
<url-pattern>*.xlsx</url-pattern>
</filter-mapping>
```

```
<filter-mapping>
<filter-name>adfdiExcelDownload</filter-name>
<url-pattern>*.xls</url-pattern>
</filter-mapping>
...
<filter-mapping>
<filter-name>ADFLibraryFilter</filter-name>
<url-pattern>/*</url-pattern>
<dispatcher>FORWARD</dispatcher>
<dispatcher>REQUEST</dispatcher>
</filter-mapping>
```

For more information about web.xml, see *Oracle Fusion Middleware Desktop Integration Developer's Guide for Oracle Application Development Framework*.

30.1.5 How to Deploy the Web Application You Created in Step 1

For an example of how to do this, see ["Task 5: Deploy the ADF Task Flow"](#) on page 30-21.

30.1.6 How to Install Microsoft Excel

Install Microsoft Excel by following the appropriate Microsoft documentation.

30.1.7 How to Install the Oracle Oracle ADF-Desktop Integration Plug-in

To do this, follow the steps in ["Task 4: Prepare the Excel Workbook"](#) on page 30-17:

30.1.8 How to Specify the User Interface Controls and Create the Excel Workbook

For instructions see ["Task 4: Prepare the Excel Workbook"](#) on page 30-17.

30.2 Attaching Excel Workbooks to Human Task Workflow Email Notifications

As an alternative to using the worklist application, you can attach an Excel workbook with task details as part of a Human Task workflow email notification. In this case, the user receives an email about a new task. This email has an Excel workbook attached, and, when the user opens the attachment, she is directed to a login page--similar to that for the worklist application. The Excel workbook includes such task details as task ID, payload, and so on. Buttons correspond to the actions the user can perform, and clicking one of them invokes the corresponding BPEL process.

30.2.1 Enabling Attachment of Excel Workbooks to Human Task Workflow Email Notifications

To enable this functionality, do the following:

1. In Oracle JDeveloper, create an ADF task flow that corresponds to a particular Human Task activity in a BPEL process.
2. Modify the settings in the ADF-DI-enabled Excel sheet to point to the server on which the task flow is deployed, then saves this Excel sheet as part of the .war file packaged for the ADF task flow. The steps for doing these things are covered in ["Example: Attaching an Excel Workbook to Email Notifications"](#) on page 5. Later,

you will use the page definition files generated in "[How to Create a Dummy JSF Page](#)" on page 30-2

Note: Packaging the Excel workbook with the ADF task flow assumes that there is a one-to-one correspondence between the ADF task flow and the Excel sheet used for a workflow.

3. Enable the ADF task flow project for desktop integration and deploys it to the server.

30.2.2 What Happens During Runtime When You Enable Attachment of Excel Workbooks to Human Task Workflow Email Notifications

Note the following end-user experience during runtime:

- A user receives an email notification regarding a new task, with the Excel attachment. When the attachment is opened, the user is directed to a login page and prompted to enter username and password. This login page is similar to the login page for worklist application.
- The Excel workbook loads up with the task details—for example, task identifier, payload. There are buttons corresponding to actions the user can take. Clicking one of these buttons starts the BPEL process in which the task is a step.

Note the following runtime behaviors:

- The Excel workbook is added as an attachment only when the flag “include task attachments” for the corresponding task is set to `true`.
- Before adding the Excel workbook as an attachment, runtime verifies that a digital signature is not enabled for the workflow.
- When the ADF task flow is deployed to the server, such data as the hostname and port number of the task flow URI is registered in the database.
- When an email notification is created, runtime retrieves from the database the hostname and port number of the application server and the context root of the task flow application. It uses this information to find the Excel workbook, `workflow_name.xls`.

30.2.3 Example: Attaching an Excel Workbook to Email Notifications

In general, you configure this integration by doing the following tasks:

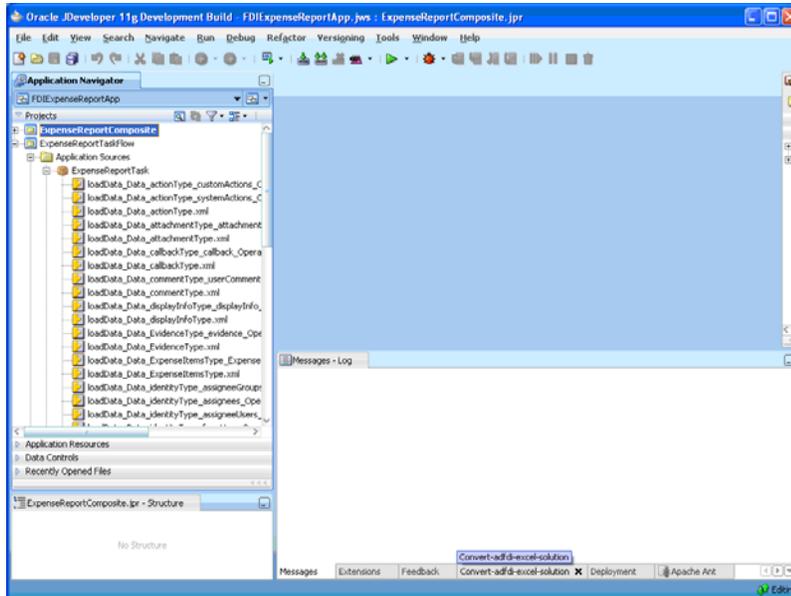
- [Task 1: Enable the ADF Task Flow Project with Oracle ADF-DI Capabilities](#)
- [Task 2: Set up Authentication](#)
- [Task 3: Create a Valid Page Definition File to Be Used in the Excel Workbook](#)
- [Task 4: Prepare the Excel Workbook](#)
- [Task 5: Deploy the ADF Task Flow](#)
- [Task 6: Test the Deployed Application](#)

30.2.3.1 Task 1: Enable the ADF Task Flow Project with Oracle ADF-DI Capabilities

In this task, you configure the web application to work with Oracle ADF-DI.

1. Create an ADF task flow project based on a Human Task. This creates a data control corresponding to the task, and .xml files corresponding to the task's structure. [Figure 30-1](#) shows JDeveloper with a sample project open.

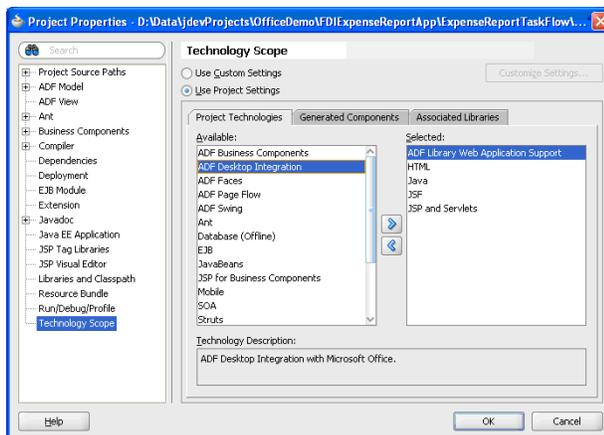
Figure 30-1 Oracle JDeveloper with a Sample Project Open



2. Add Oracle ADF Desktop Integration to the project by following the instructions in "[How to Add Desktop Integration to Your Oracle JDeveloper Project](#)" on page 30-2.

[Figure 30-2](#) illustrates the Oracle JDeveloper Project Properties dialog when you are adding Oracle ADF Desktop Integration to your project.

Figure 30-2 Oracle JDeveloper Project Properties Dialog

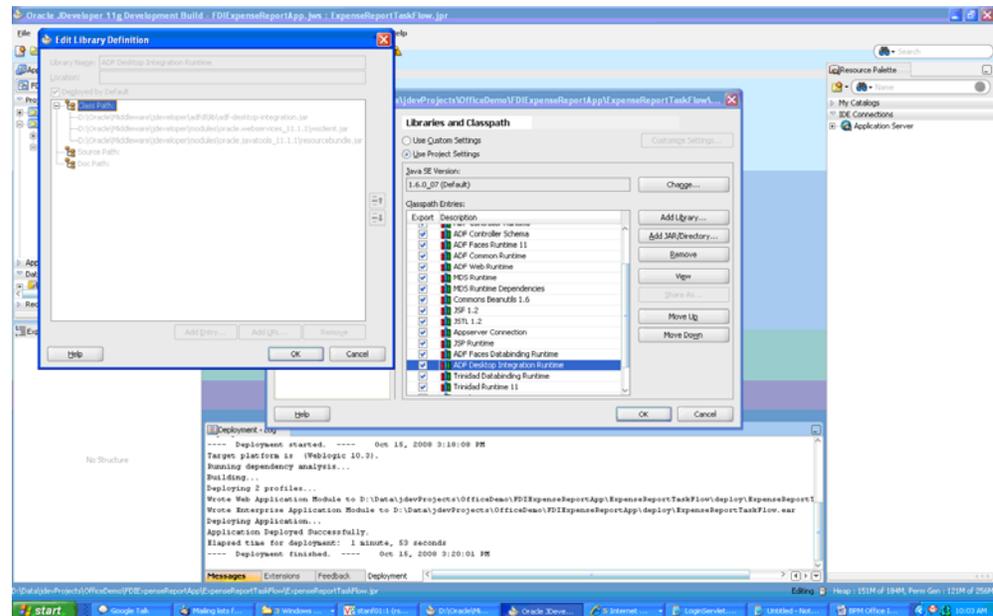


3. When the technology scopes mentioned in Step 2 are added to the project, verify that the necessary events have occurred. To do this:
 - a. In the Application Navigator, right-click the Project.
 - b. Click Project Properties, then select Libraries and Classpath.

- c. Confirm that the entry ADF Desktop Integration Runtime exists and is checked.
- d. Select this library and click View.
- e. Confirm that the library references `wsclient.jar` and `adf-desktop-integration.jar` in its classpath.

Figure 30-3 shows the sequence of dialogs you used to make this verification.

Figure 30-3 Oracle JDeveloper: Verifying the Technology Scope of a Project



4. Confirm that the project's deployment descriptor—namely, `web.xml`—is modified to include the following entries:
 - A servlet named `adfdiRemote`
 - A filter named `adfdiExcelDownload`
 - A MIME mapping for Excel files

The previous list is not exhaustive. Adding “ADF Desktop Integration” and “ADF Library Web Application Support” to the project makes other changes to `web.xml`. Here is a sample snippet of the deployment descriptor:

```
<context-param>
  <param-name>javax.faces.STATE_SAVING_METHOD</param-name>
  <param-value>client</param-value>
</context-param>
```

```
<context-param>
  <description>...</description>
  <param-name>org.apache.myfaces.trinidad.CHECK_FILE_MODIFICATION
</param-name>
  <param-value>>false</param-value>
</context-param>
```

```
<context-param>
  <description>Whether the 'Generated by...' comment at the bottom of ADF
  Faces HTML pages should contain version number information.</description>
  <param-name>oracle.adf.view.rich.versionString.HIDDEN</param-name>
  <param-value>>false</param-value>
```

```
</context-param>
<filter>
    <filter-name>trinidad</filter-name>
    <filter-class>org.apache.myfaces.trinidad.webapp.TrinidadFilter
</filter-class>
</filter>
<filter>
    <filter-name>ADFLibraryFilter</filter-name>
    <filter-class>oracle.adf.library.webapp.LibraryFilter
</filter-class>
</filter>
<filter>
    <filter-name>adfBindings</filter-name>
    <filter-class>oracle.adf.model.servlet.ADFBindingFilter
</filter-class>
</filter>
<filter>
    <filter-name>adfdiExcelDownload</filter-name>
    <filter-class>
oracle.adf.desktopintegration.filter.DIExcelDownloadFilter
</filter-class>
</filter>
<filter-mapping>
    <filter-name>trinidad</filter-name>
    <servlet-name>Faces Servlet</servlet-name>
    <dispatcher>FORWARD</dispatcher>
    <dispatcher>REQUEST</dispatcher>
</filter-mapping>
<filter-mapping>
    <filter-name>adfBindings</filter-name>
    <servlet-name>Faces Servlet</servlet-name>
    <dispatcher>FORWARD</dispatcher>
    <dispatcher>REQUEST</dispatcher>
</filter-mapping>
<filter-mapping>
    <filter-name>trinidad</filter-name>
    <servlet-name>adfdiRemote</servlet-name>
</filter-mapping>
<filter-mapping>
    <filter-name>adfBindings</filter-name>
    <servlet-name>adfdiRemote</servlet-name>
</filter-mapping>
<filter-mapping>
    <filter-name>adfdiExcelDownload</filter-name>
    <url-pattern>*.xlsx</url-pattern>
</filter-mapping>
<filter-mapping>
    <filter-name>adfdiExcelDownload</filter-name>
    <url-pattern>*.xlsm</url-pattern>
</filter-mapping>
<filter-mapping>
    <filter-name>ADFLibraryFilter</filter-name>
    <url-pattern>/*</url-pattern>
    <dispatcher>FORWARD</dispatcher>
    <dispatcher>REQUEST</dispatcher>
</filter-mapping>
<servlet>
    <servlet-name>Faces Servlet</servlet-name>
    <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
```

```

        <load-on-startup>1</load-on-startup>
</servlet>
<servlet>
    <servlet-name>resources</servlet-name>

<servlet-class>org.apache.myfaces.trinidad.webapp.ResourceServlet</servlet-class>
</servlet>
<servlet>
    <servlet-name>adflibResources</servlet-name>

    <servlet-class>oracle.adf.library.webapp.ResourceServlet</servlet-class>
</servlet>
<servlet>
    <servlet-name>adfdiRemote</servlet-name>

<servlet-class>oracle.adf.desktopintegration.servlet.DIRemoteServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>Faces Servlet</servlet-name>
    <url-pattern>/faces/*</url-pattern>
</servlet-mapping>
<servlet-mapping>
    <servlet-name>resources</servlet-name>
    <url-pattern>/adf/*</url-pattern>
</servlet-mapping>
<servlet-mapping>
    <servlet-name>resources</servlet-name>
    <url-pattern>/afr/*</url-pattern>
</servlet-mapping>
<servlet-mapping>
    <servlet-name>adflibResources</servlet-name>
    <url-pattern>/adflib/*</url-pattern>
</servlet-mapping>
<servlet-mapping>
    <servlet-name>adfdiRemote</servlet-name>
    <url-pattern>/adfdiRemoteServlet</url-pattern>
</servlet-mapping>
<session-config>
    <session-timeout>35</session-timeout>
</session-config>
<mime-mapping>
    <extension>html</extension>
    <mime-type>text/html</mime-type>
</mime-mapping>
<mime-mapping>
    <extension>txt</extension>
    <mime-type>text/plain</mime-type>
</mime-mapping>
<mime-mapping>
    <extension>xlsx</extension>

<mime-type>application/vnd.openxmlformats-officedocument.spreadsheetml.sheet</mime-type>
</mime-mapping>
<mime-mapping>
    <extension>xlsm</extension>
    <mime-type>application/vnd.ms-excel.sheet.macroEnabled.12</mime-type>
</mime-mapping>

```

5. Add the following `<auth-filter>` entry to `weblogic.xml`.

```
<weblogic-web-app>

<auth-filter>oracle.bpel.services.workflow.client.worklist.util.FDIFilter</auth-
-filter>
.
.
</weblogic-web-app>
```

6. Click **Save All**. Right click the project and click **Rebuild**. Make sure there are no compilation errors and the build completes successfully.

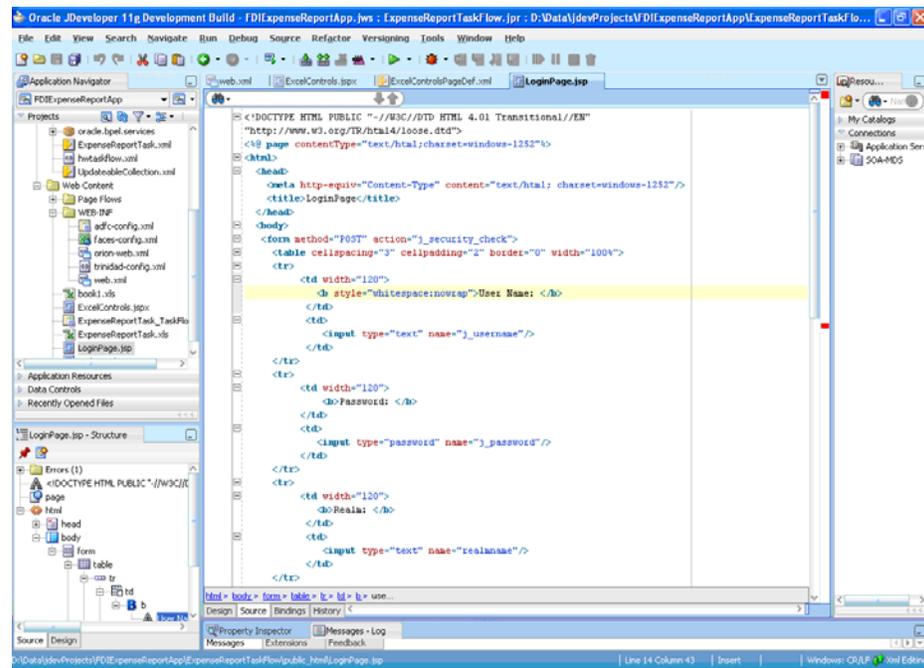
The web application is now configured to work with Oracle ADF-DI.

30.2.3.2 Task 2: Set up Authentication

This task is required to add Oracle ADF-Desktop Integration to create a web session for an Excel workbook.

1. Add ADF security to your project. To do this:
 - a. From the Application menus, then Secure, then Configure ADF Security.
 - b. Select ADF Authentication.
 - c. Click **Finish**.
2. Create a login page for the application. To do this:
 - a. From the directory `ExpenseReportTaskFlow\public_html\` copy the file `LoginPage.jsp` to the directory `project_home\public_html`.
 - b. Refresh the view in Oracle JDeveloper.
 - c. Verify that the file `LoginPage.jsp` is visible. It should look like what is illustrated in [Figure 30-4](#).

Figure 30-4 Oracle JDeveloper: Login.jsp File



- Once you have added ADF security, confirm that the following entries are added to the file web.xml. If some entries are missing, add them manually. Note that form authentication, using the login page created in Step 2 on page 30-10, is used.

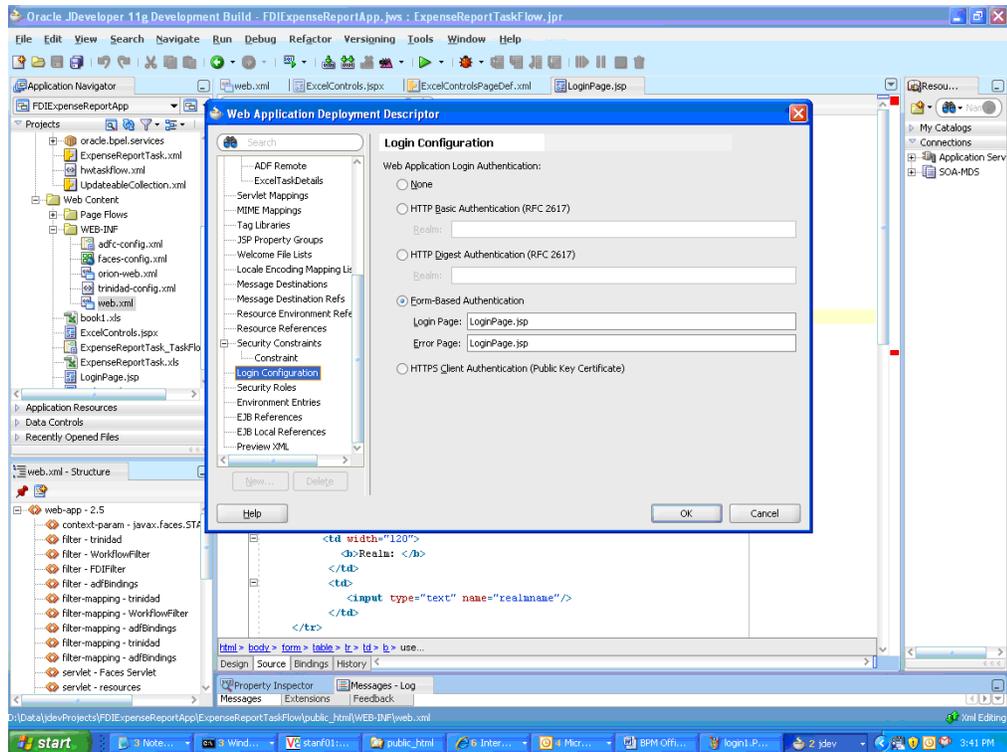
```

<security-constraint>
    <web-resource-collection>
        <web-resource-name>allPages</web-resource-name>
        <url-pattern>/</url-pattern>
    </web-resource-collection>
    <auth-constraint>
        <role-name>Administrators</role-name>
    </auth-constraint>
</security-constraint>
<security-constraint>
    <web-resource-collection>
        <web-resource-name>adfAuthentication</web-resource-name>
        <url-pattern>/adfAuthentication</url-pattern>
    </web-resource-collection>
    <auth-constraint>
        <role-name>Administrators</role-name>
    </auth-constraint>
</security-constraint>
<login-config>
    <auth-method>FORM</auth-method>
    <realm-name>jazn.com</realm-name>
    <form-login-config>
        <form-login-page>/LoginPage.jsp</form-login-page>
        <form-error-page>/LoginPage.jsp</form-error-page>
    </form-login-config>
</login-config>
<security-role>
    <role-name>Administrators</role-name>
</security-role>

```

Figure 30-5 shows how these entries appear graphically in the Application Deployment Descriptor dialog.

Figure 30-5 Oracle JDeveloper: Application Deployment Descriptor



4. For every logical security role added in `web.xml`, make a corresponding entry in `weblogic.xml` as follows:

```
<weblogic-web-app>

<auth-filter>oracle.bpel.services.workflow.client.worklist.util.FDIAuthFilter</auth-filter>

  <security-role-assignment>
    <role-name>Administrators</role-name>
    <principal-name>fmwadmin</principal-name>
    <principal-name>users</principal-name>
  </security-role-assignment>
  :
</weblogic-web-app>
```

5. Click **Save All**.

The ADF Task Flow web application is now configured for login capability that can be used by the Excel workbook.

30.2.3.3 Task 3: Create a Valid Page Definition File to Be Used in the Excel Workbook

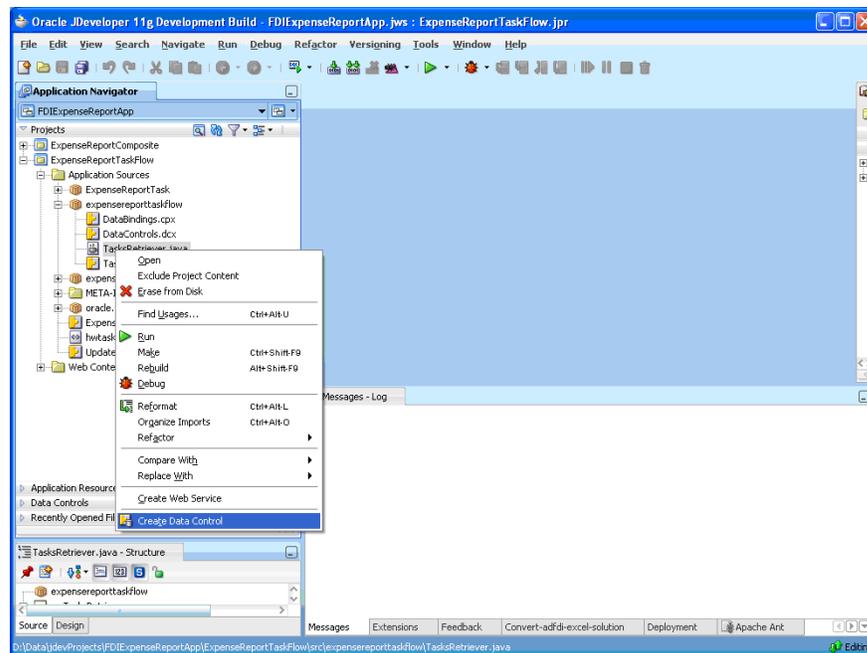
The page definition file `ExcelControlsPageDef.xml` is used to create the Excel workbook. To do this:

1. Create a new Java class by following these steps:

- a. Select Technologies, then select General, then select Simple Files, then select Java Class.
 - b. Specify details as follows:
 - Name: TaskRetrievers
 - Package: (leave it as default)
 - Extends: oracle.bpel.services.workflow.client.worklist.excel.TasksRetriever (click Browse to select this class)

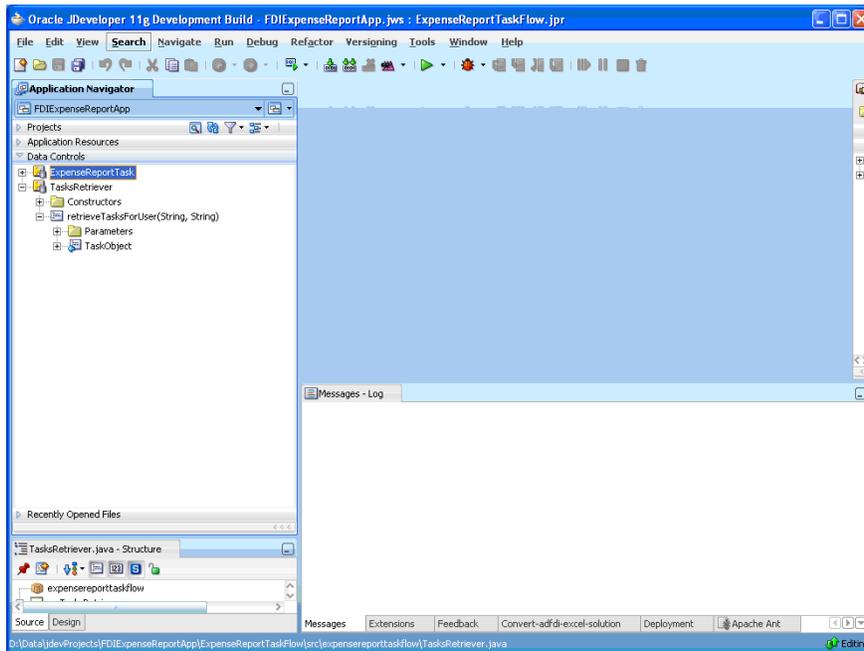
This creates a new Java class <default-package>.TasksRetriever.
2. Create a data control for this newly created Java class. This data control provides access to an API that retrieves all assigned tasks for a user. [Figure 30–6](#) shows the menu involved in creating the data control.

Figure 30–6 Oracle JDeveloper: Creating a Data Control



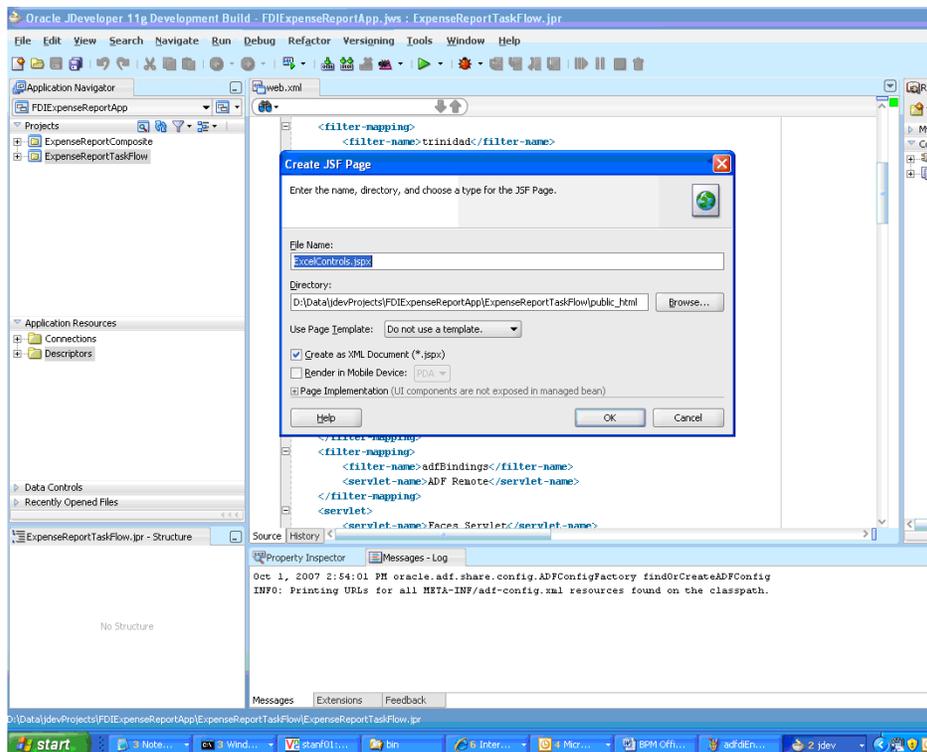
3. Verify that the newly created Data Control TasksRetriever is visible in the Data Control palette in the lower portion of the Application Navigator. [Figure 30–7](#) shows the Application Navigator with the Data Control palette expanded.

Figure 30–7 Oracle JDeveloper: Application Navigator with Data Control Palette Expanded



4. Create a new JSF JSP page--namely, `ExcelControls.jspx`. This generates a page definition that can be used by ADF-DI while authoring the Excel document.

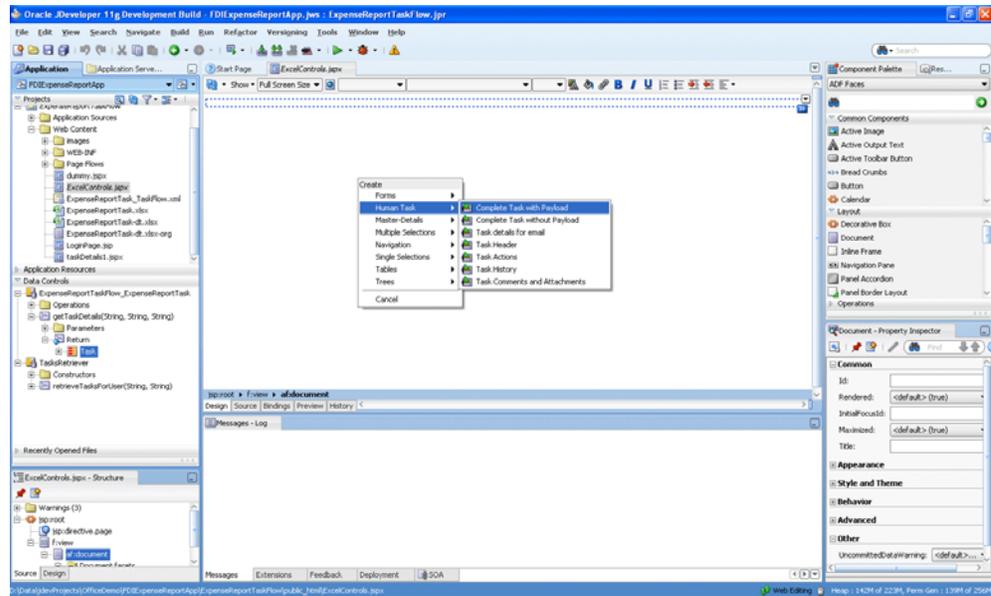
Figure 30–8 Oracle JDeveloper: Creating a JSF JSP Page



5. Drag and drop the task node from the Data Controls palette to `ExcelControls.jspx`. Select Human Task, then select Complete task with

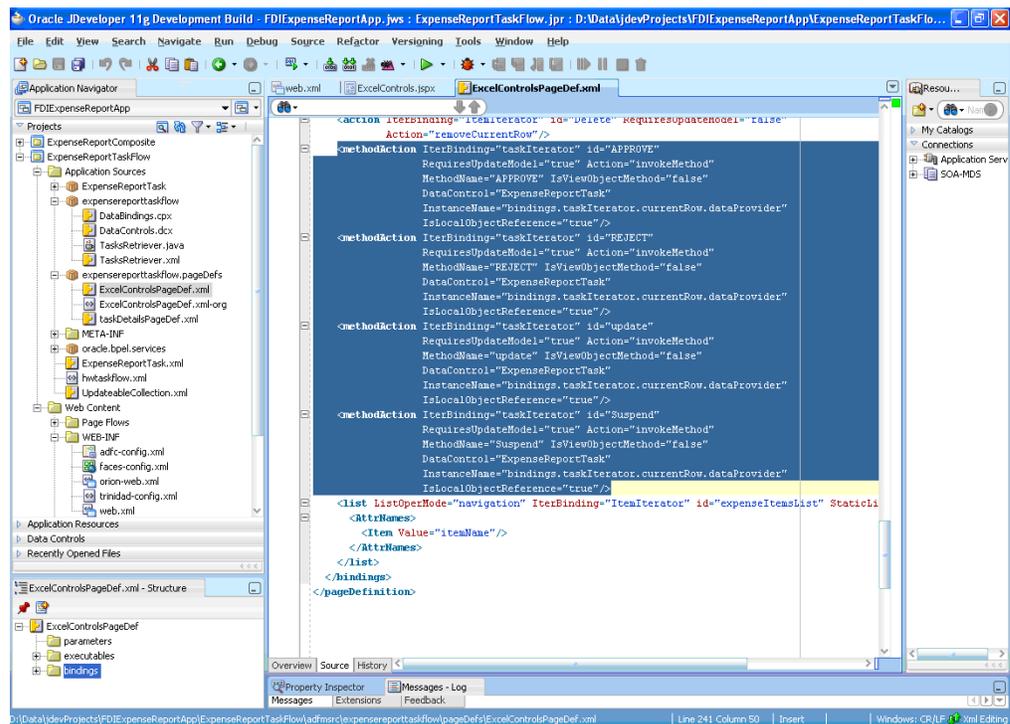
payload. Figure 30–9 illustrates the sequence of menus you use. Click OK on windows that pop up.

Figure 30–9 Oracle JDeveloper: Creating an ADF Read-Only Form



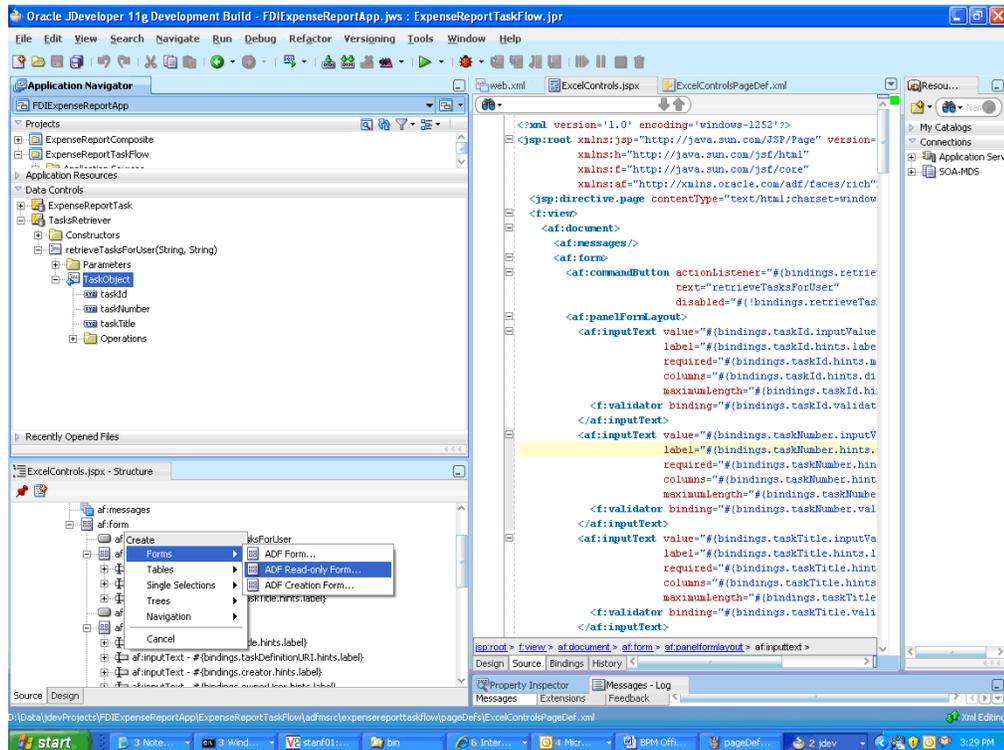
6. Drag and drop one or more task actions to the .jspx file. In this example, as illustrated in Figure 30–10, the actions 'Approve', 'Reject', 'update' and 'Suspend' are added to create the entries in the page definition.

Figure 30–10 Oracle JDeveloper: Configuring the Page Definition File



7. Drag and drop the `retrieveTasksForUser()` method from the Data Controls palette (expand the node `TasksRetriever`) to `ExcelControls.jspx`. For now, click **OK** on the Edit Action Binding dialog. This creates a binding in `ExcelControlsPageDef.xml` to extract all assigned tasks for the logged-in user.
8. Drag and drop `TaskObject` from the Data Control palette to `ExcelControls.jspx` to create an ADF Read Only Form. Verify that a corresponding `<methodIterator>` executable and `<attributeValues>` bindings are created in `ExcelControlsPageDef.xml`.

Figure 30–11 Oracle JDeveloper: Page Definition File



9. Depending on the number of task details to be exposed in the Excel workbook, drag and drop as many ADF controls as needed. In this example, we expose only as many task details as needed to develop a minimally operational workbook.
10. Create a list binding in `ExcelControlsPageDef.xml` that can be used to create a list of assigned tasks in the Excel workbook. Add the following entry to the `<bindings>` element in the page definition.

```
<list ListOperMode="navigation"
      IterBinding="retrieveTasksForUserIterator" id="retrievedTaskList"
      StaticList="false">
  <AttrNames>
    <Item Value="taskNumber" />
  </AttrNames>
</list>
```

11. Similarly add the following list binding in `ExcelControlsPageDef.xml` that can be later used to create a list of an updatable table of expense items in the Excel workbook.

```

<list ListOperMode="navigation" IterBinding="ItemIterator"
  id="expenseItemsList" StaticList="false">
  <AttrNames>
    <Item Value="itemName"/>
  </AttrNames>
</list>

```

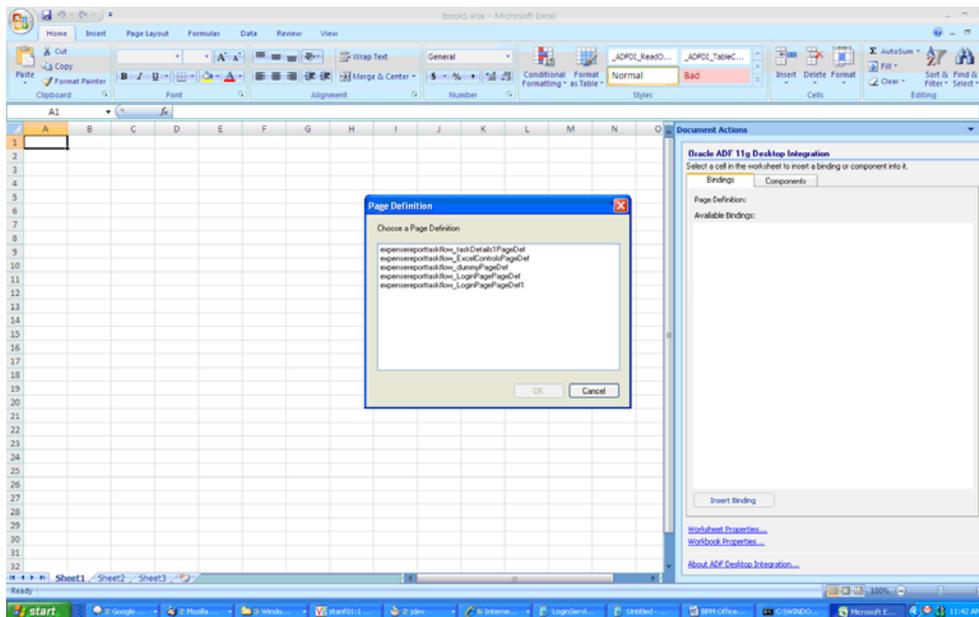
12. Click **Save All**. Right click the project and click **Rebuild**. Make sure that there are no compilation errors and the build completes successfully.

30.2.3.4 Task 4: Prepare the Excel Workbook

To author the Excel workbook, follow these steps:

1. For information about desktop requirements for running the ADF-DI solution, read Section 3.1 of *Oracle Fusion Middleware Desktop Integration Developer's Guide for Oracle Application Development Framework*.
2. Configure security for Excel. To do this:
 - a. Open Excel.
 - b. Click the **Microsoft Office** button, then click **Excel Options**.
 - c. Click the Trust Center tab, then click Trust Center Settings.
 - d. Click the Macro Settings tab, then click the checkbox labeled Trust Access to the VBA project object model.
 - e. Click **OK**.
3. Run the setup tool that comes with the Oracle ADF-DI module. The setup tool is stored in the following folder: `JDEV_HOME\jdeveloper\adfdi\bin\excel\client`
4. Create a new Excel workbook in the directory `project_home\public_html\`. Click **View**, then click **Refresh**. This displays the Excel workbook in Oracle JDeveloper.
5. Run the conversion command on the Excel workbook. The Oracle ADF-DI module stores the conversion tool, `convert-adfdi-excel-solution.exe`, in `oracle_jdeveloper_home\jdeveloper\adfdi\bin\excel\convert`. To convert the Excel workbook, execute the following command:
`convert-adfdi-excel-solution.exe <workbook.xlsx> -attach`
 The Excel workbook is now enabled to use the Oracle ADF-DI framework.
6. Open the Excel workbook and choose a page definition. In this use case, the page definition is `expensereporttaskflow_ExcelControlsPageDef`.

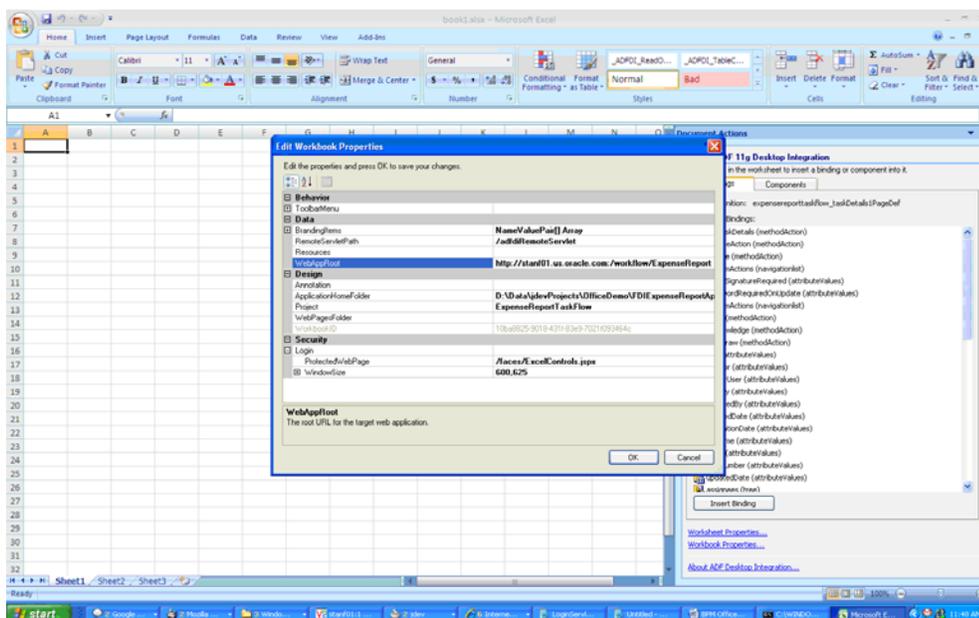
Figure 30–12 Excel: Page Definition Dialog



7. In the Document Actions pane, select Workbook Properties.
8. Specify ProtectedWebPage: `http://application_server:port/workflow/application_name/faces/app/logon`. (Note that this URL is protected and will trigger form authentication. See [Section 30.2.3.2, "Task 2: Set up Authentication"](#) on page 30-10).
Specify WebAppRoot: `http://application_server:port/workflow/application_name`. Click OK.

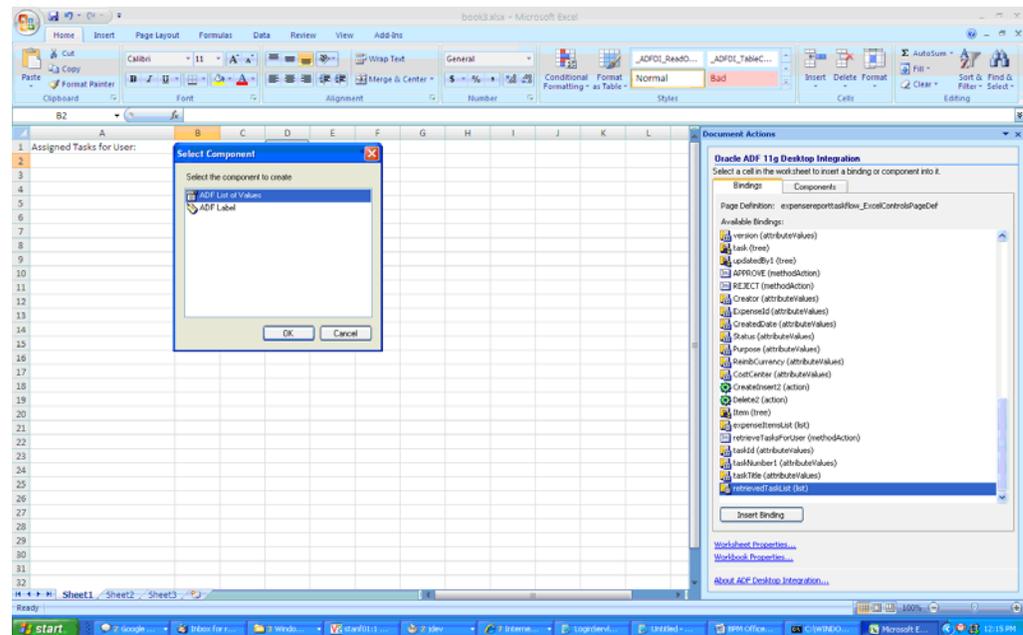
See Also: Section C-2 of *Oracle Fusion Middleware Desktop Integration Developer's Guide for Oracle Application Development Framework*

Figure 30–13 Excel: Setting WebAppRoot



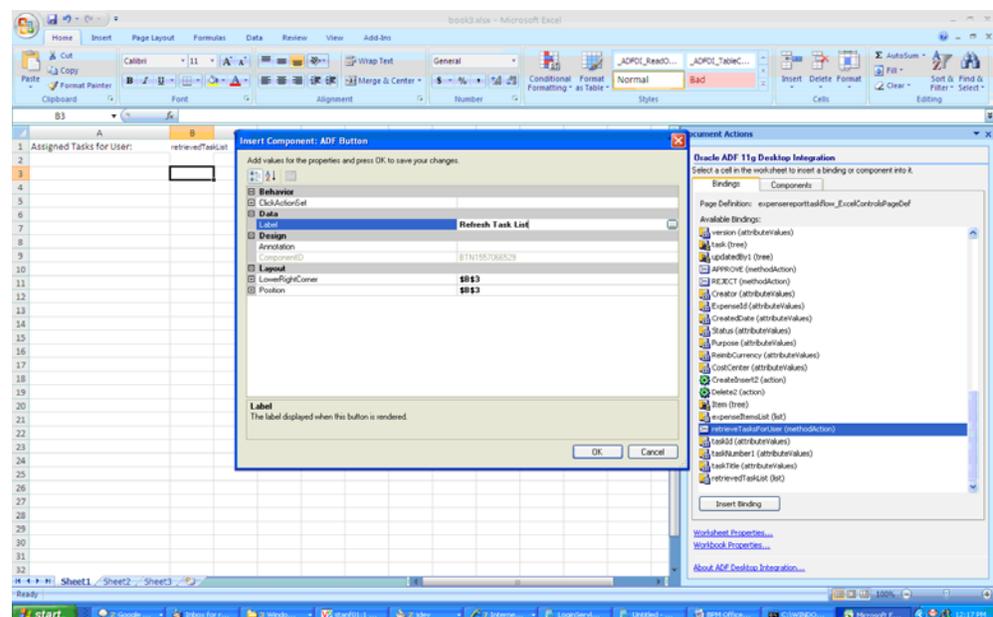
- From the Document Actions pane, insert ADF Bindings to create the corresponding fields in the Excel workbook. For further details on specific components refer to the *Oracle Fusion Middleware Desktop Integration Developer's Guide for Oracle Application Development Framework*. For instance, insert binding `retrievedTaskList` to create a list of values.

Figure 30–14 Excel: Creating a List of Values



- Insert a `methodAction` binding to create a button in Excel.

Figure 30–15 Excel: Inserting a methodAction Binding

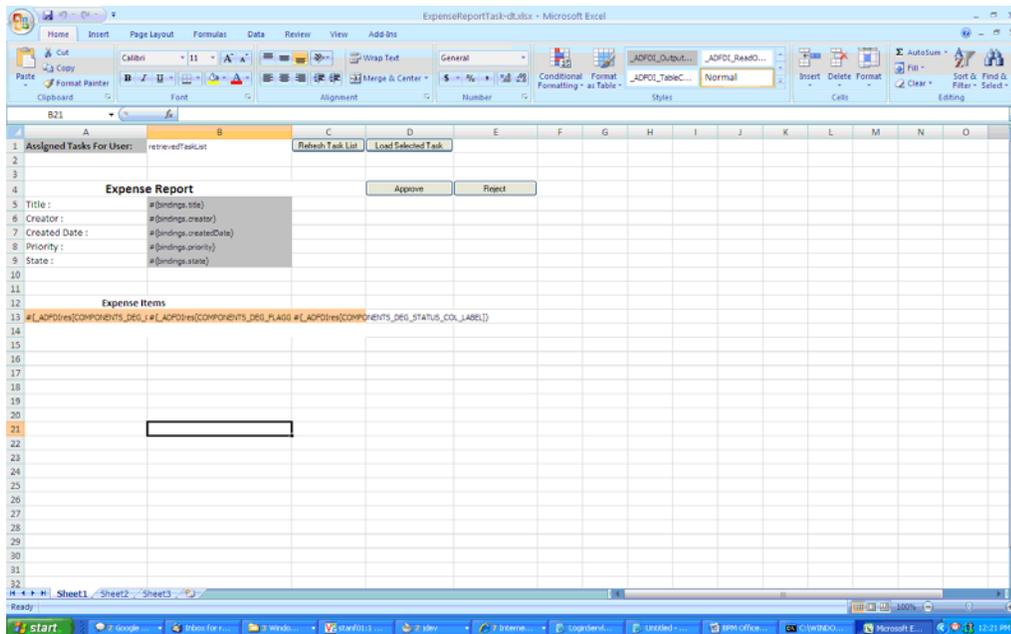


11. Insert a tree binding to create an ADF Table component. A Table component is an updatable table of records in Excel. For instance, the list binding `expenseItemsList` is a candidate for a Table component.

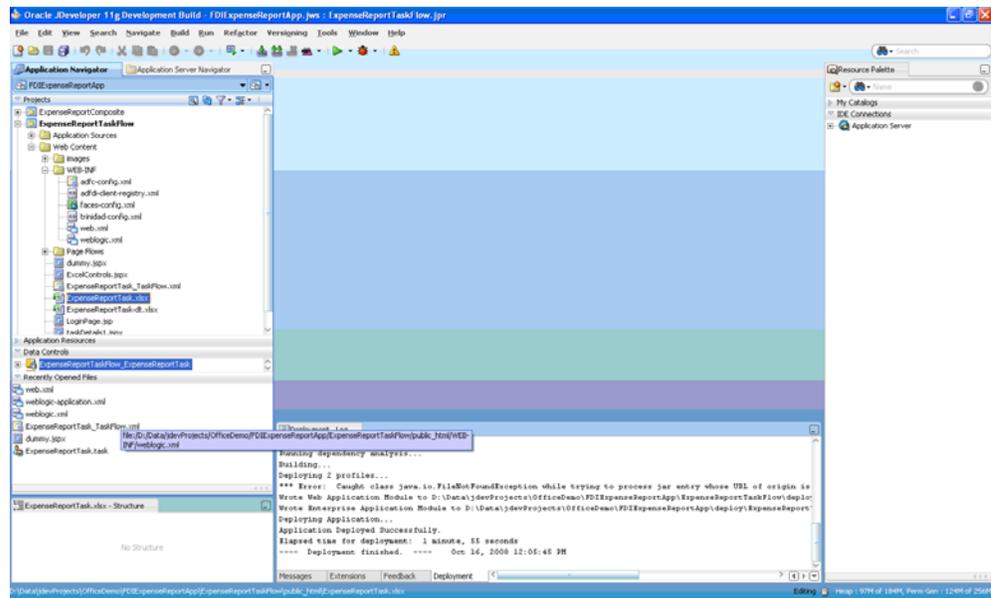
See Also: *Oracle Fusion Middleware Desktop Integration Developer's Guide for Oracle Application Development Framework* for further information about creating and modifying a Table component.

A completed Excel workbook for an expense report application looks something like what you see in [Figure 30-16](#):

Figure 30-16 *Excel Workbook Integrated with Oracle ADF-DI*



12. Publish the workbook by following these steps:
 - a. On the toolbar, click **Publish**. The Publish Workbook dialog appears.
 - b. In the File name field, specify the name as `workflow_name.xls`. The workflow name is the value of the element `WorkflowName` specified in `project_home\adfmsrc\hwtaskflow.xml`. In this example, the name of the published Excel workbook is `ExpenseReportTask.xls`.
13. In Oracle JDeveloper, click **View**, then click **Refresh**. Verify that the published workbook is visible under Web Content as illustrated in [Figure 30-17](#).

Figure 30–17 Oracle JDeveloper: Verifying Workbook Under WebContent

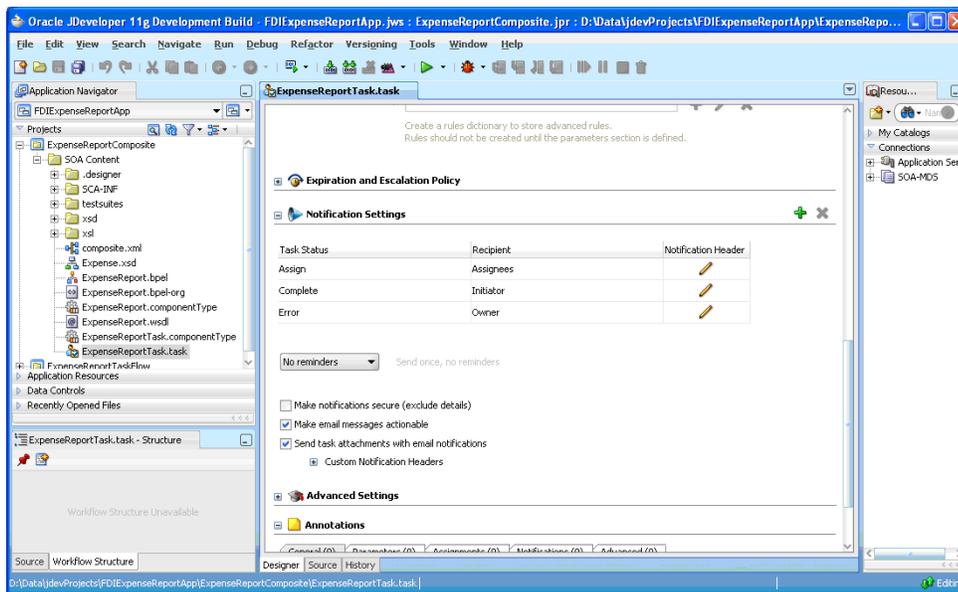
14. Click **Save All**. The ADF Task Flow is now ready for deployment.

30.2.3.5 Task 5: Deploy the ADF Task Flow

To deploy the ADF Task Flow, follow these steps:

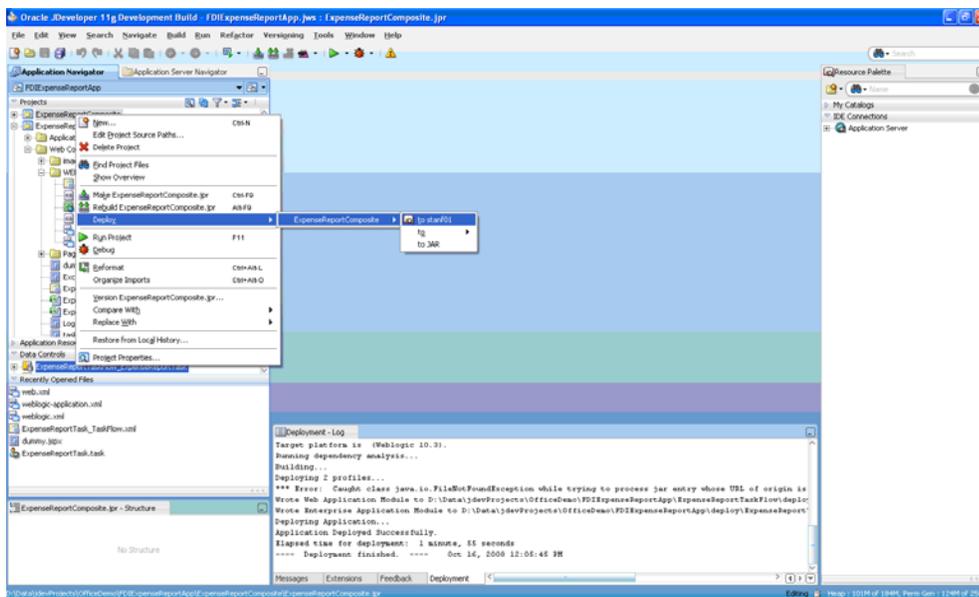
1. For the Excel workbook to be sent as an attachment when a task is assigned, you must configure the corresponding task in the SCA Composite. To do this:
 - a. In Oracle JDeveloper, open the SCA Composite Project that corresponds to the ADF Task Flow.
 - b. Open the `.task` file.
 - c. Verify that the item labeled Send task attachments with email notifications is checked.

Figure 30–18 Oracle JDeveloper: Configuring SCA Composite Task



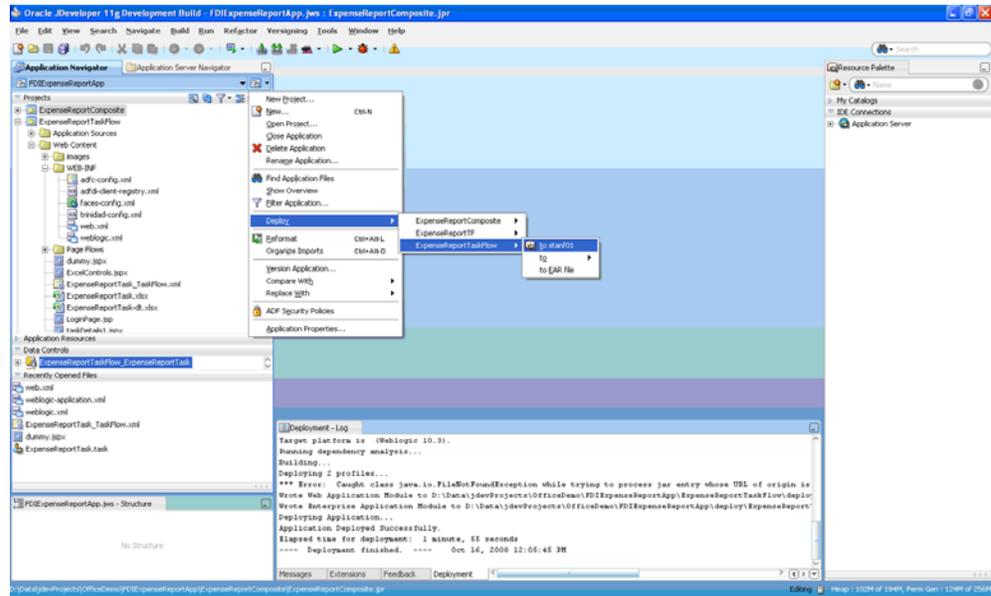
2. Deploy the application. To do this, right-click the SOA Composite, select **Deploy**, select the composite application name, and then select the application server. [Figure 30–19](#) shows the sequence of menu selections.

Figure 30–19 Oracle JDeveloper: Deploying the Application



3. Deploy the ADF Task Flow. To do this: In the Application Navigator, expand Projects, and select the application. Then select Deploy, then application_TaskFlow (In this example, the application task flow is ExpenseReportTaskFlow), then select the application server. [Figure 30–20](#) shows what the sequence of menus may look like.

Figure 30–20 Oracle JDeveloper: Menu Sequence when Deploying an ADF Task Flow



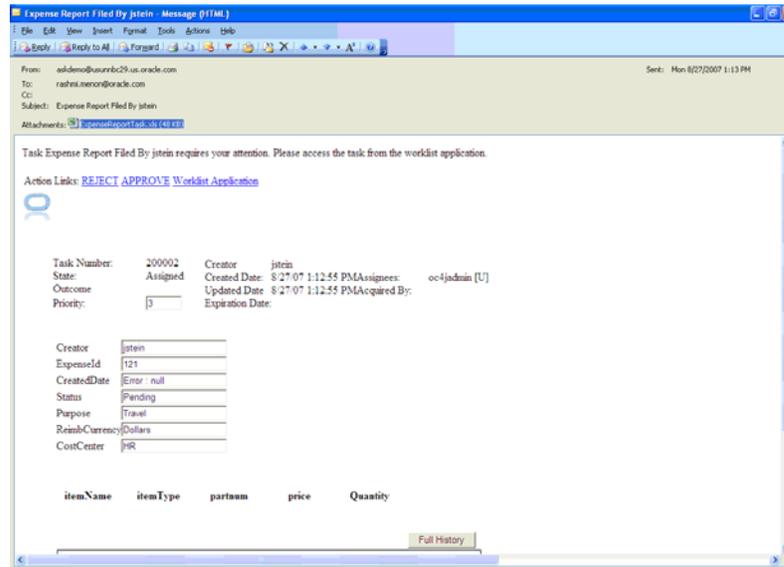
At this point, the ADF Task Flow is successfully deployed.

30.2.3.6 Task 6: Test the Deployed Application

To test the deployed application, follow these steps:

1. Invoke the deployed SOA Composite and verify that the assignee receives the Excel workbook as part of the email notification.

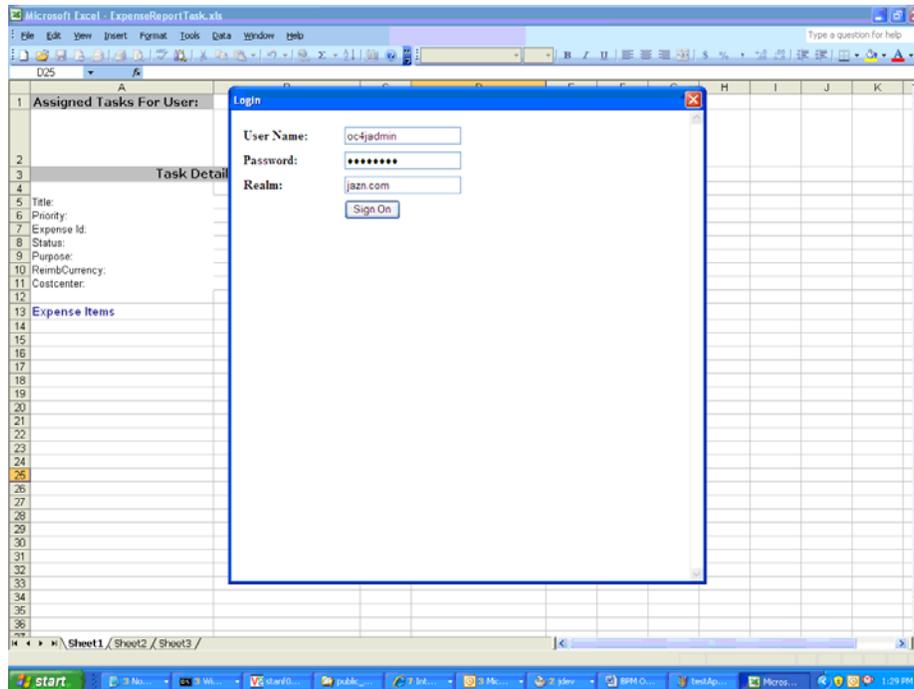
Figure 30–21 Excel Workbook Attached to an Email



Note: To successfully open and execute the workbook, the user's desktop machine should have the correct security policy and needs to run the `caspol` command to grant trust to the client assemblies hosted on the network share.

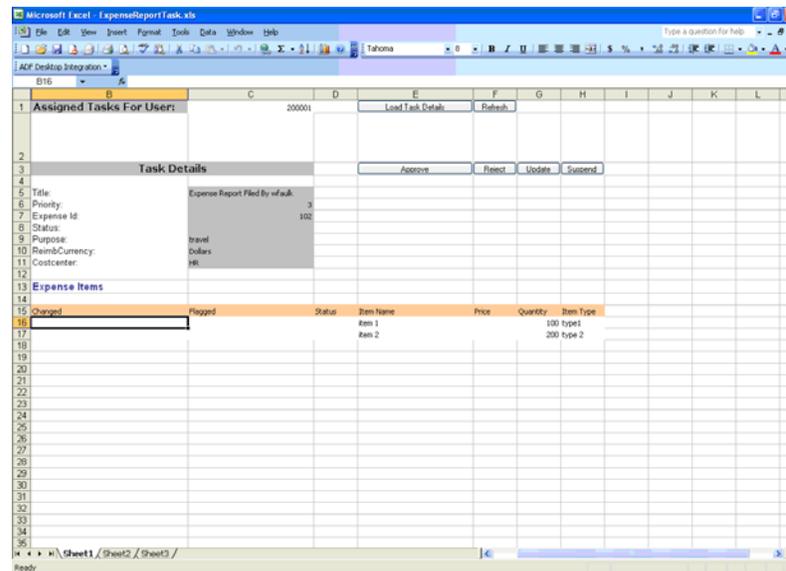
2. Open the Excel workbook. You are directed to a login page (This is `LoginPage.jsp` from "How to Create a Dummy JSF Page" on page 30-2). Enter your security credentials.

Figure 30-22 Desktop-Integrated Excel Workbook: Login Page



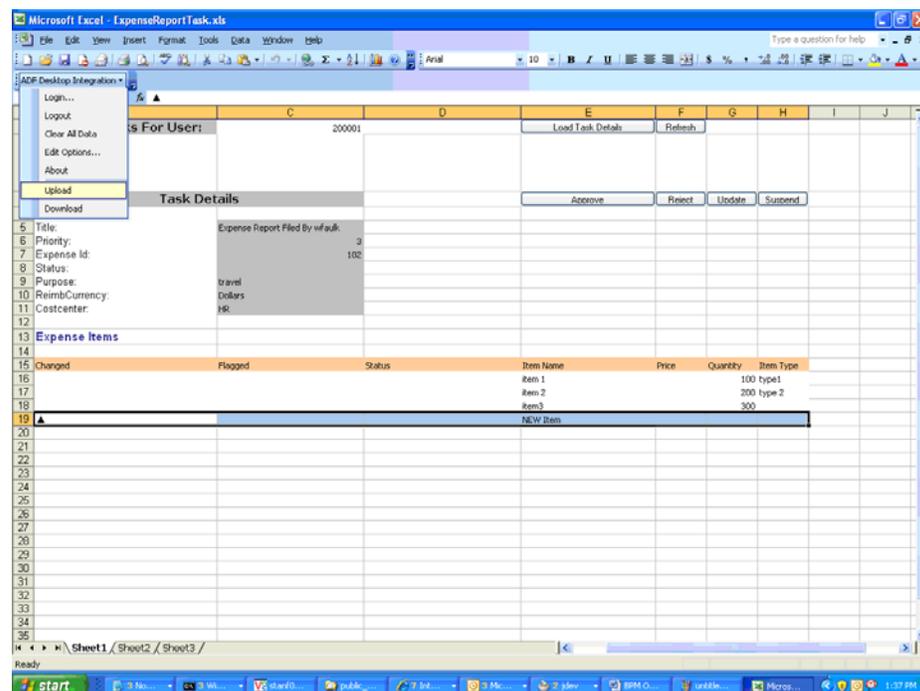
3. Examine the workbook to verify the following:
 - All the assigned tasks for the logged-in user are retrieved in the Excel workbook.

Figure 30–23 ADF Desktop-Integrated Excel Workbook with Assigned Tasks



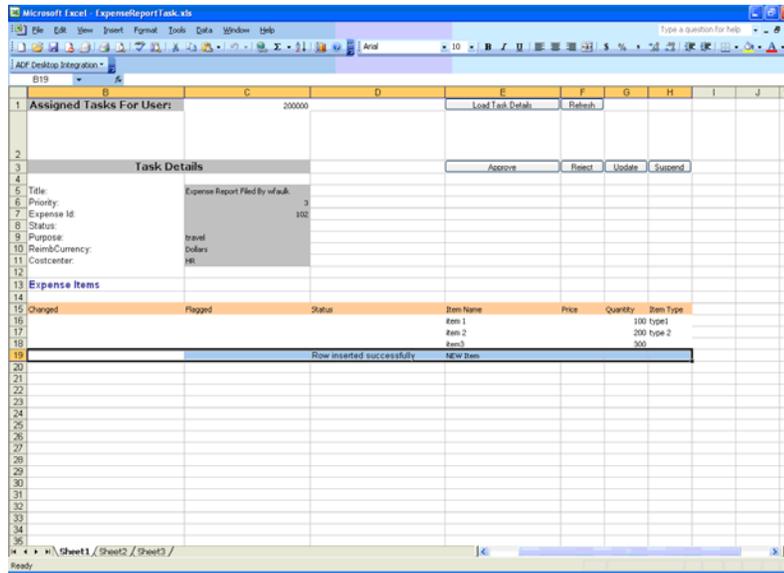
- You can navigate to the needed task from the list of assigned tasks and update it as required. For instance, as illustrated in Figure 30–24, in the Expense Report application, you can upload new expense items.

Figure 30–24 ADF Desktop-Integrated Excel Workbook Uploading New Items



- The Status column in the workbook indicates if the upload was successful. Also, you can perform actions on the task by clicking **Approve**, **Reject**, **Update**, or **Suspend**.

Figure 30–25 ADF Desktop-Integrated Excel Workbook



Part VI

Using Oracle Business Activity Monitoring

This part describes Oracle Business Activity Monitoring.

This part contains the following chapters:

- [Chapter 31, "Creating Oracle BAM Data Objects"](#)
- [Chapter 32, "Integrating Oracle BAM with SOA Composite Applications"](#)
- [Chapter 33, "Creating Oracle BAM Enterprise Message Sources"](#)
- [Chapter 34, "Using Oracle Data Integrator With Oracle BAM"](#)
- [Chapter 35, "Creating External Data Sources"](#)
- [Chapter 36, "Using Oracle BAM Web Services"](#)
- [Chapter 37, "Creating Oracle BAM Alerts"](#)
- [Chapter 38, "Using ICommand"](#)

Creating Oracle BAM Data Objects

This chapter contains the information needed to create and manage data objects, including assigning permissions, managing folders, creating security filters, and adding dimensions and hierarchies.

This chapter contains the following topics:

- [Section 31.2, "Defining Data Objects"](#)
- [Section 31.3, "Creating Permissions on Data Objects"](#)
- [Section 31.4, "Viewing Existing Data Objects"](#)
- [Section 31.5, "Using Data Object Folders"](#)
- [Section 31.6, "Creating Security Filters"](#)
- [Section 31.7, "Creating Dimensions"](#)
- [Section 31.8, "Renaming and Moving Data Objects"](#)
- [Section 31.9, "Creating Indexes"](#)
- [Section 31.10, "Clearing Data Objects"](#)
- [Section 31.11, "Deleting Data Objects"](#)
- [Section 31.2.6, "What You May Need to Know About System Data Objects"](#)

31.1 Introduction to Creating Data Objects

Data objects are tables that store raw data in the database. Each data object has a specific layout which can be a combination of data fields, lookup fields, and calculated fields.

The data objects you define are based on the types of data available from Enterprise Message Sources (EMS) that you can define in Oracle BAM Architect. You must define columns in the data object. The data object contains no data when you create it. You must load or stream data into data objects using the technologies discussed in the following sections:

- [Chapter 32, "Integrating Oracle BAM with SOA Composite Applications"](#)
- [Chapter 33, "Creating Oracle BAM Enterprise Message Sources"](#)
- [Chapter 34, "Using Oracle Data Integrator With Oracle BAM"](#)
- [Chapter 35, "Creating External Data Sources"](#)
- [Chapter 36, "Using Oracle BAM Web Services"](#)

Data objects can also be accessed by Oracle BAM alerts. See [Chapter 37, "Creating Oracle BAM Alerts"](#) for more information.

WARNING: Do not read or manipulate data directly in the database. All access to data must be done using Oracle BAM Architect or the Oracle BAM Active Data Cache API.

31.2 Defining Data Objects

31.2.1 How to Define a Data Object

To define a data object:

1. Select **Data Objects** from the Oracle BAM Architect function list.
2. Click **Create Data Object**.
3. Enter a name for the data object.
4. Enter the path to the location in the folder tree in which to store the data object. Click **Browse** to use the **Select a Folder** dialog box.
5. Optionally, enter a description of the data object.
6. If this data object is loaded from an External Data Source (EDS) select the **External Data Source** checkbox and configure the following:
 - Select an **External Data Source** from the list. EDS definitions are configured on the External Data Sources screen. See [Chapter 35, "Creating External Data Sources"](#) for more information.
 - Select the **External Table Name**.

Note: Only the tables that belong to the user are shown when a data object is created on an EDS.

Creating a data object with multiple time stamp fields on an EDS is not supported.

7. Add columns to the data object using the **Add a field** or **Add one or more lookup fields** options.

See [Section 31.2.2, "How to Add Columns to a Data Object"](#) and [Section 31.2.3, "How to Add Lookup Columns to a Data Object"](#) for more information.

8. Click **Create Data Object** when you are finished adding columns or lookup columns.

31.2.2 How to Add Columns to a Data Object

To add columns to a data object:

1. In a data object you are creating or editing, click **Add a field**.
2. Specify the column name, data type, maximum size (scale for decimal columns), whether it is nullable, whether it is public, and tip text.

If you are adding a column in a data object based on an External Data Source you must also supply the **External field name**.

The data types include:

- **String.** Text columns containing a sequence of characters.
A string with a max size greater than 0 and less than or equal to 2000 becomes an Oracle database data type VARCHAR field. If the max size is less than zero or greater than 2000 the string field is stored as a CLOB. To get a CLOB field, just define a string field with a max size greater than 2000.
- **Integer.** Numeric columns containing whole numbers from -2,147,483,648 to 2,147,483,648.
- **Float.** Double-precision floating point numbers.
The Oracle BAM Float type does not map to the Oracle database Float type. Oracle BAM Float truncates numeric data that has very high precision. If you do not want to see loss of precision use the Oracle BAM Decimal type (NUMBER in Oracle database) with the scale you want.
- **Decimal.** Numbers including decimal points with scale number defined. The number is stored as a string which uses a character for each digit in the value.
The Oracle BAM Decimal data type is stored as a NUMBER (38, X) in the Oracle database. The first argument, 38, is the precision, and this is hard-coded. The second argument, X, is the scale, and you can adjust this value. The scale value cannot be greater than 38.
- **Boolean.** Boolean columns with true or false values.
- **Auto-incrementing integer.** Automatically incremented integer column.
- **DateTime.** Dates and times combined as a real number.
- **Timestamp.** Date time stamp generated to milliseconds. A data object can contain only one time stamp field. See [Section 31.2.5, "How to Add Time Stamp Columns to a Data Object"](#) for more information.
A DateTime field is stored as an Oracle database data type DATE. A Timestamp field is stored as an Oracle database data type TIMESTAMP(6). Depending on how the Timestamp field is populated, Oracle BAM may fill in the time stamp value for you. For instance, in Oracle BAM Architect you cannot specify the value for Timestamp when adding a row, but if the value for Timestamp is specified in an ICommand import file, the specified value is added as the value of Timestamp instead of the current time.
- **Calculated.** Calculated columns are generated by an expression and saved as another data type. See [Section 31.2.4, "How to Add Calculated Columns to a Data Object"](#) for more information.

Keep adding columns using **Add a field** and **Add one or more lookup fields** until all the required columns are listed. Click **Remove** to remove a column in the data object.

3. Click **Save changes**.

31.2.3 How to Add Lookup Columns to a Data Object

You can add lookup columns to a data object. This performs lookups on key columns in a specified data object to return columns to the current data object. You can match multiple columns and return multiple lookup columns.

To add a lookup column to a data object:

1. In a data object you are creating or editing, click **Add one or more lookup fields**.
The Define Lookup Field dialog box opens.
2. Select the data object to use for the lookup.
3. Select the lookup columns from the data object. You can select one or more columns by holding down the Shift or Control key when selecting. Selecting multiple columns creates multiple lookup columns in the data object. These are the columns you want to return.
4. Select the column to match from the lookup data object.
5. Select the column to match from the current data object. You must have previously created other columns in this data object so that you have a column to select.
6. Click **Add**.
The matched column names are displayed in the list. You can click **Remove** to remove any matched pairs you create.
7. You can repeat steps 4 through 6 to create multiple matched columns. This is also known as a composite key.
8. Click **OK** to save your changes and close the dialog box.

The new lookup columns are added to the data object. Click **Modify Lookup Field** in **Layout > Edit Layout page** to make changes to a lookup column. Multiple selection of return columns is possible when defining a new lookup but not when modifying an existing one.

You can click **Remove** to remove any lookups you create.

Note: Oracle Business Activity Monitoring supports two types of schema models: unrelated tables or star schemas. Any other kind of schema that does not conform to these models may result in performance issues or deadlocks. Snowflake dimensions (daisy-chained lookups) are not supported.

Supported:

Table 1 (with no lookups to any other tables)
Table 1 > Lookup > Table 2

Not supported:

Table 1 > Lookup > Table 2 > Lookup > Table 3

31.2.4 How to Add Calculated Columns to a Data Object

When creating calculated columns in a data object you can use operators and expression functions, combined with column names, to produce a new column.

[Table 31-1](#) Describes the operators you can use to build calculated columns.

The *Oracle Fusion Middleware User's Guide for Oracle Business Activity Monitoring* provides the syntax and examples for expressions you can use in a calculated column.

Table 31–1 Operators Used in Calculated Columns

Operator	Function
+ (plus sign)	Add
- (minus sign)	Subtract
* (asterisk)	Multiply
/ (slash)	Divide
% (percent sign)	Modulus
() (parentheses)	Parentheses determine the order of operations
&& (double ampersand)	Logical AND
!= (exclamation point and equal sign)	Logical NOT
(double pipe)	Logical OR For example <pre>if ((CallbackClientTime == NULL) (ReceiveInputTime == NULL)) then (-1) else (CallbackClientTime-ReceiveInputTime)</pre>
== (double equal sign)	Equality
= (equal sign)	Assignment

Column names containing any special characters, such as the operators listed in [Table 31–1](#) double quotation marks, or spaces, must be surrounded with curly braces {}. If column names contain only numbers, letters and underscores and begin with a letter or underscore they do not need curly braces. For example, if the column name is **Sales+Costs**, the correct way to enter this in a calculation is {Sales+Costs}.

Double quotation marks must be escaped with another set of double quotation marks if used inside double quotation marks. For example, `Length(" "Hello World, " "I said")`.

WARNING: If you enter a calculated column with incorrect syntax in a data object, you could lose the data object definition.

31.2.5 How to Add Time Stamp Columns to a Data Object

You can create a date time stamp column generated to milliseconds by selecting the **Timestamp** data type. This column in the data object must be empty when the data object is populated by the Oracle BAM ADC so that the time stamp data can be created.

31.2.6 What You May Need to Know About System Data Objects

The System data objects folder contains data objects used to run Oracle Business Activity Monitoring. You should not make any changes to these data objects, except for the following:

- **Custom Parameters** lets you define global parameters for Action Buttons.
- **Action Form Templates** lets you define HTML forms for Action Form views.
- **Chart Themes** lets you add or change color themes for view formatting.

- **Matrix Themes** lets you add or change color themes for the Matrix view.
- **Util Templates** lets you define templates that are used by Action Form views to transform content.

For more information about matrix and color themes, Action Buttons, and Action Forms see *Oracle Fusion Middleware User's Guide for Oracle Business Activity Monitoring*.

31.2.7 What You May Need to Know About Oracle Data Integrator Data Objects

If you install the integration files for Oracle BAM and Oracle Data Integrator, three data objects are created in Oracle BAM Architect: Context, Scenarios and Variables in the `/System/ODI/` folder. These data objects should not be deleted from Oracle BAM Architect, and their configuration should not be changed.

31.3 Creating Permissions on Data Objects

You can add permissions for users and groups on data objects. When users have at least a read permissions on a data object they can choose the data object when creating reports.

31.3.1 How to Create Permissions on a Data Object

To add permissions on a data object:

1. Select **Data Objects** from the Oracle BAM Architect function list.
2. Select the data object.

The general information for the data object is displayed in the right frame.

3. Click **Permissions**.
4. Click **Edit Permissions**.

Alternatively you can copy permissions from another data object. See [Section 31.3.3, "How to Copy Permissions from Other Data Objects"](#) for more information.

5. Click the **Restrict access to Data Object to certain users or groups** checkbox.
The list of users and groups and permissions is displayed.
6. You can choose to display the following by choosing an option:
 - Show all users and groups
 - Show only users and groups with permissions
 - Show users only
 - Show groups only

7. You can set permissions for the entire list by clicking the buttons at the top of the list.

The permissions are Read, Update, and Delete. You can set permissions for individual users or groups in the list by clicking the checkbox in the permission column that is next to the user or group name.

Note: Delete and Update permissions are not effective unless a user is also granted the Read permission.

Members of the Administrator role have all permissions to all data objects, and their permissions cannot be edited.

8. After indicating the permissions with selected checkboxes, click **Save changes**.
A message is displayed to confirm that your changes are saved.
9. Click **Continue** to display the actions for the data object.

31.3.2 How to Add a Group of Users

Users assigned to the Administrator role have access to all data objects. The Administrator role overrides the data object permissions.

To add a group to the list:

1. Click **Add a group to the list**.
2. Type the Windows group name in the field. The group must previously exist as a domain group.
3. Click **OK**.

The group is added to the list.

31.3.3 How to Copy Permissions from Other Data Objects

You can copy the permissions from another data object and then make additional changes to the permissions before saving.

In Oracle BAM Architect for a data object, click **Permissions** and then click **Copy from**. Select the data object that contains the permissions to copy and click **OK**. You can edit the copied permissions and click **Save changes**.

To copy permissions from another data object:

1. Select **Data Objects** from the Oracle BAM Architect function list.
2. Click the data object to add a security filter to.
The general information for the data object is displayed in the right frame.
3. Click **Permissions**.
4. Click **Copy from**.
The Choose Data Object dialog box opens.
5. Select the data object that contains the permissions to copy and click **OK**.
6. If the data object previously had no permissions assigned, select the **Restrict access to Data Object** checkbox.
7. You can edit the copied permissions or add a group to the list.
8. Click **Save changes**.

31.4 Viewing Existing Data Objects

This section describes how to view information about data objects.

31.4.1 How to View Data Object General Information

The general information of a data object displays the owner, when it was created, when it was last modified, and the row count.

To view the general information of a data object:

- Click the data object in the list.

If you are currently viewing the layout or contents of a data object, click *General*.

The general information is displayed in the right frame. It contains the following information:

- **Created.** Date and time the data object was created.
- **Last modified.** Date and time the data object was last modified.
- **Row count.** Number of rows of data in the data object.
- **Location.** Location of the data object.
- **Type.** Type of the data object.
- **Data Object ID.** The ID used to identify the data object. This is based on the name although the ID is used throughout the system so that you can edit the name without affecting any dependencies.

Note: If the row count is over 500,000 rows, an approximate row count is displayed in the General information for increased performance purposes. The approximate row count is accurate within 5-10% of the actual count. If you want to view an exact row count instead of the approximation, click Show exact count. The exact count is displayed. This could take a few minutes if the data object has millions of rows.

31.4.2 How to View Data Object Layouts

The layout describes the columns in a data object. The columns are described by name, column ID, data type, maximum length allowed, scale, nullable, public, calculated, text tip, and lookup.

To view the layout of a data object:

1. Select the data object.
2. The general information is displayed in the right frame.
3. Click **Layout**.

The layout information is displayed in the right frame. It contains the following information:

- **Field name.** Name of the column.
- **Field ID.** Generated by the system.
- **External name.** External column name from the External Data Source (only appears in data objects based on External Data Sources).
- **Field type.** Data type of the column.
- **Max length.** Maximum number of characters allowed in column value.
- **Scale.** Number of digits on the right side of the decimal point.

- **Nullable.** Whether the data type can contain null values.
- **Public.** This setting determines if the column is available in Oracle BAM Active Studio to use in a report. If the box is unchecked, the column does not appear in Oracle BAM Active Studio. This is useful for including columns for calculations in data objects that should not appear in reports.
- **Lookup.** Displays specifics of a lookup column.
- **Calculated.** Displays the expression of a calculated column.
- **Tip Text.** Helpful information about the column.

31.4.3 How to View Data Object Contents

You can view the rows of data stored in a data object by viewing the data object contents. You can also edit the contents of the data object.

To view the contents of a data object:

1. Select the data object.

The general information is displayed in the right frame.

2. Click **Contents**.

The first 100 rows of the data object display in the right frame.

(To change this default, update the `Architect_Content_PageSize` property. See *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite* for information.)

3. Click **Next**, **Previous**, **First**, and **Last** to go to other sets of rows.

Rows are listed with a Row ID column. Displaying only Row ID provides faster paging for large data objects. Row IDs are assigned one time in each row and maintain a continuous row count when you clear and reload a data object.

You can click **Show row numbers** to display an additional column containing a current row count starting at 1. Click **No row numbers** to hide the row count column again.

4. Click **Refresh** to get the latest available contents.

31.5 Using Data Object Folders

You can organize data objects by creating folders and subfolders for them. When you create a folder for data objects, you can assign permissions by associating users and actions with the folder.

31.5.1 How to Create Folders

You can create new folders for organizing data objects. Then you can move or create data objects into separate folders for different purposes or users. After creating folders, you can set folder permissions to limit which users can view the data objects it contains.

To create a new folder:

1. Select **Data Objects** from the Oracle BAM Architect function list.

The current data object folders display in a tree hierarchy.

2. Click **Create subfolder**.
A field for naming the new folder is displayed.
3. Enter a name for the folder and click **Create folder**.
The folder is created as a subfolder under the Data Objects folder and a message is displayed confirming that the new folder was created.
4. Click **Continue** to view the folder.

31.5.2 How to Open Folders

To open a folder:

1. Expand the tree of folders by clicking the + (plus sign) next to the Data Objects folder.

The System subfolders contain data objects for running Oracle Business Activity Monitoring. For more information about these data objects see [Section 31.2.6, "What You May Need to Know About System Data Objects."](#)

2. Click the link next to a folder to open it.

The folder is opened, and the data objects in the folder are shown in the list underneath the folder tree. The general properties for the folder display in the right frame and the following links apply to the current folder:

View. Displays the general properties of this folder such as name, date created, date last modified, user who last modified it. View is selected when you first click a folder.

Create subfolder. Creates another folder within the selected folder.

Delete. Removes the selected folder and all the data objects it contains.

Rename. Changes the folder name.

Move. Moves this folder to a new location, for example, as a subfolder under another folder.

Permissions. Sets permissions on this folder.

Create Data Object. Creates a data object in this folder.

31.5.3 How to Set Folder Permissions

When you create a folder, you can set permissions on it so that other users can access the data objects contained in the folder.

To set permissions on a folder:

1. In the Data Objects folder, select the folder to change permissions on.
2. Click **Permissions**.
3. Click **Edit permissions**.
4. Select the **Restrict access to Data Object to certain users or groups** checkbox.
The list of users and groups and permissions is displayed.
5. You can choose to display the following by selecting a radio button:
 - Show all users and groups

- Show only users and groups with permissions
 - Show users only
 - Show groups only
6. You can set permissions for the entire list by clicking the column headers at the top of the list.

The permissions are Read, Update, and Delete. You can set permissions for individual users or groups in the list by selecting the checkbox in the permission column that is next to the user or group name.

Note: Delete and Update permissions are not effective unless a user is also granted the Read permission.

7. After indicating the permissions with selected checkboxes, click **Save changes**.
A message is displayed to confirm that your changes are saved.
8. Click **Continue** to display the actions for the data object.

To add a group to the list:

1. Click the **Add a group to the list** link.
2. Type the Windows group name in the field. The group must previously exist as a domain group.
3. Click **OK**.

The group is added to the list.

31.5.4 How to Move Folders

To move a folder:

1. Select the folder to move.
2. Click **Move**.
3. Click **Browse** to select the new location for the folder.
4. Click **OK** to close the dialog box.
5. Click **Move folder**.

The folder is moved.

6. Click **Continue**.

31.5.5 How to Rename Folders

To rename a folder:

1. Select the folder to rename.
2. Click **Rename**.
3. Enter a new name and click **Rename folder**.

The folder is renamed. You must assign unique folder names within a containing folder.

4. Click **Continue**.

31.5.6 How to Delete Folders

When you delete a folder, you also delete all of the data objects in the folder.

To delete a folder:

1. Select the folder to delete.
2. Click **Delete**.

A message is displayed to confirm deletion of the folder and all of its contents.

3. Click **OK**.

The folder is deleted.

4. Click **Continue**.

31.6 Creating Security Filters

You can add security filters to data objects so that only specific users can view specific rows in the data object. This can be useful when working with data objects that contain sensitive or confidential information that is not intended for all report designers or report viewers.

Security filters perform a lookup using another data object, referred to as a security data object, containing user names or group names. Before you can add a security filter, you must create a security data object containing the user names or group names and the value in the column to allow for each user name or each group name. Security data objects cannot contain null values.

If the user has a view open, and you change that user's security filter, it does not effect the currently open view. If the user reopens that view, it has the new security filter settings applied. Security filter settings are used to construct the query behind the view at view construction time, so changes to a security filter are not seen by previously created views.

31.6.1 How to Create a Security Filter

To add a security filter to a data object:

1. Select **Data Objects** from the Oracle BAM Architect function list.
2. Select the data object to add a security filter to.

The general information for the data object is displayed in the right frame.

3. Select **Security Filters**.

If the data object includes security filters, the filter name is displayed and you can expand and view the information.

4. Click **Add filter**.

The fields for defining the security filter display.

5. Enter the following information:

Name of this Security Filter. Type a name for this filter.

Security Data Object. Select the name of the security data object containing the mapped columns.

Type of identification. Select either By user or By group from the dropdown list. The security data object must include either domain or local users or groups mapped to values in the identification column.

Identification column in Security Data Object. Select the name of the column for containing user names or group names.

Match column in Security Data Object. Select the column to match in the security data object.

Match column in this Data Object. Select the name of the column to match in this data object.

6. Click Add.

For example, to add a security filter to the following data object, you need a security data object containing Region information to perform the security lookup.

Sample data object:

User	Region	Sales
John Smith	1	\$55,000
Bob Wright	1	\$43,000
Betty Reid	2	\$38,000

Security data object:

Login	Region ID
DomainName\jsmith	1
DomainName\jsmith	2
DomainName\bwright	1
DomainName\breid	2

When the **bwright** account views a report that accesses the data object with a security filter applied based on Region ID and Region, it is only able to access information for jsmith and bwright. It is not able to view the breid information because it is not able to view data for the same region. On the other hand, the jsmith account is set up to view data in both region 1 and 2.

31.6.2 How to Copy Security Filters from Other Data Objects

You can copy security filters from another data object and apply them to the data object you are editing.

To copy security filters from another data object:

1. Select **Data Objects** from the Oracle BAM Architect function list.
2. Select the data object to add a security filter to.

The general information for the data object is displayed in the right frame.

3. Select **Security Filters**.

If the data object includes security filters, the filter name is displayed and you can expand and view the information.

4. Click **Copy from**.
The Choose Data Object dialog box opens.
5. Select the data object that contains the security filters to copy and click **OK**.
6. You can make changes to the security filters by viewing the filter details and clicking **Edit**.
7. Click **Save**.

31.7 Creating Dimensions

In Oracle BAM Architect, you can add dimensions to data objects to define drill paths for charts in Oracle BAM Active Studio. Dimensions contain columns in a hierarchy. When a hierarchy is selected in chart, the end user can drill down and up the hierarchy of information. When a user drills down in a chart, they can view data at more and more detailed levels.

Hierarchies are an attribute of a dimension in a data object. Multiple dimensions can be created in each data object. Each column in a data object can belong to one dimension only. You can create and edit multiple, independent hierarchies.

To use hierarchies as drill paths in charts, the report designer must select the hierarchy to use as the drill path. To create a dimension, you must select multiple columns to save as a dimension. Then you organize the columns into a hierarchy.

The following is a sample dimension and hierarchy:

Dimension	Hierarchy
Sales	Category
	Brand
	Description

31.7.1 How to Create a Dimension

To add a dimension and hierarchy:

1. Select **Data Objects** from the Oracle BAM Architect function list.
2. Select the data object to add a dimension to.
The general information for the data object is displayed in the right frame.
3. Select **Dimensions**.
4. Click **Add a new dimension**.
5. Enter a dimension name.
6. Enter a description for the dimension. A description is required for drilling configuration.
7. Select the column names to include in the dimension. An example is Sales, Category, Brand, and Description.
The column names are moved from the Data Objects Fields list to the Dimension Fields list to show that they are selected.
8. Click **Save**.
9. Click **Continue**.

The new dimension is listed. You must still define a hierarchy for the columns.

10. Click **Add new hierarchy**.
11. Enter a hierarchy name.
12. Enter a description for the hierarchy.
13. Select the column names to define as attributes for the dimension. An example is Sales remains in the Dimension Field list, and you click Category, Brand, and Description to arrange them in a general to more specific order. The order that you click the columns is the order that they are listed in the Hierarchy Field list. Arrange the more general grouping column at the top of the Hierarchy Fields list and the most granular column at the bottom of the Hierarchy Fields list.
14. Click **Save**.
15. Click **Continue**.

The new hierarchy is listed. You can edit or remove hierarchies and dimensions by clicking the links. You can also continue defining multiple hierarchies for the dimension or add new dimensions to the data object.

31.7.2 How to Create a Time Dimension

If your dimension contains a time date data type column, you can select the time levels to include in the hierarchy.

To select time levels:

1. In a dimension containing a time date data type column, add a hierarchy.
2. Select the time date data type column. If you are editing existing time levels, click **Edit Time Levels**.

The Time Levels Definition dialog box opens.

3. Click the levels to include in the hierarchy. The levels include:
 - **Year.** Year in a four digit number.
 - **Quarter.** Quarter of four quarters starting with quarter one representing January, February, March.
 - **Month.** Months one through 12, starting with January.
 - **Week of the Year.** Numbers for each week starting with January 1st.
 - **Day of the Year.** Numbers for each day of the year starting with January 1st.
 - **Day of the Month.** Numbers for each day of the month.
 - **Day of the Week.** Numbers for each day of the week, starting from Sunday to Saturday.
 - **Hour.** Numbers from one to twenty four.
 - **Minute.** Numbers from one to 60.
 - **Second.** Numbers from one to 60.
4. Click **OK** to close the dialog box.

31.8 Renaming and Moving Data Objects

You can rename and move a data object without editing or clearing the data object. If you only want to change the data object name or description, use the Rename option.

31.8.1 How to Rename a Data Object

To rename a data object:

1. Select **Data Objects** from the Oracle BAM Architect function list.
2. Select the data object to rename.

The general information for the data object is displayed in the right frame.

3. Select **Rename/Move**.
4. Enter the new name, tip text, and description for the data object.
5. Click **Save changes**.

31.8.2 How to Move a Data Object

To move a data object:

1. Select **Data Objects** from the list.
2. Select the data object to rename.

The general information for the data object is displayed in the right frame.

3. Select **Rename/Move**.
4. Click **Browse** to enter the new location for the data object.
5. Click **Save changes**.

31.9 Creating Indexes

Indexes improve performance for large data objects containing many rows. Without any indexes, accessing data requires scanning all rows in a data object. Scans are inefficient for very large data objects. Indexes can help find rows with a specified value in a column.

If the data object has an index for the columns requested, the information is found without having to look at all the data. Indexes are most useful for locating rows by values in columns, aggregating data, and sorting data.

31.9.1 How to Create an Index

You can add indexes to data objects by selecting columns to be indexed as you are creating a data object.

To add an index:

1. Select **Data Objects** from the Oracle BAM Architect function list.
2. Select the data object to add an index to.
3. Select **Indexes**.
4. Click **Add Index**.

The Add Index dialog box opens.

5. Enter a **Name** and **Description** for the index
6. Add as many columns as needed to create an index for the table.
Click a column in the list on the right to remove the column from the index.
7. Click **OK**.

The index is added and is named after the columns it contains. You can create multiple indexes. To remove an index you created, click **Remove Index** next to the Index name.

31.10 Clearing Data Objects

Clearing a data object removes the current contents without deleting the data object from the Oracle BAM ADC.

31.10.1 How to Clear a Data Object

To clear a data object:

1. Select **Data Objects** from the Oracle BAM Architect function list.
2. Select the data object to clear.
The general information for the data object is displayed in the right frame.
3. Click **Clear**.

31.11 Deleting Data Objects

When deleting data objects, you must remove referrals to the data object from reports and alerts that are using it. If the data object is in use by an active alert or report, it cannot be deleted in Oracle BAM Architect.

31.11.1 How to Delete a Data Object

To delete a data object:

1. Select **Data Objects** from the Oracle BAM Architect function list.
2. Click the data object to delete.
The general information for the data object is displayed in the right frame.
3. Click **Delete**.

Integrating Oracle BAM with SOA Composite Applications

Oracle BAM provides an adapter available in the SOA Composite Editor in Oracle JDeveloper. This chapter provides information about using the Oracle BAM Adapter and Oracle BAM sensor actions to integrate SOA composite applications with Oracle BAM.

This chapter contains the following topics:

- [Section 32.1, "Introduction to Integrating Oracle BAM with SOA Composite Applications"](#)
- [Section 32.2, "Configuring Oracle BAM Adapter"](#)
- [Section 32.3, "Creating a Design Time Connection to an Oracle BAM Server"](#)
- [Section 32.4, "Using Oracle BAM Adapter in an SOA Composite Application"](#)
- [Section 32.5, "Using Oracle BAM Adapter in a BPEL Process"](#)
- [Section 32.6, "Integrating BPEL Sensors with Oracle BAM"](#)

32.1 Introduction to Integrating Oracle BAM with SOA Composite Applications

The Oracle BAM Adapter is a Java Connector Architecture (JCA)-compliant adapter which can be used from a Java EE client to send data and events to the Oracle BAM Server. The Oracle BAM Adapter supports the following operations on Oracle BAM data objects: inserts, updates, upserts, and deletes.

The Oracle BAM Adapter can perform these operations over Remote Method Invocation (RMI) calls (if they are deployed in the same farm), direct Java object invocations (if they are deployed in the same container), or over Simple Object Access Protocol (SOAP) (if there is a fire wall between them)

Oracle BAM Adapter is configured in Oracle WebLogic Server Administration Console to provide any of these connection pools. Oracle BAM Adapter provides three mechanisms by which you can send data to Oracle BAM Active Data Cache from an SOA composite application.

The Oracle BAM Adapter supports the active Oracle BAM Server migration automatically, and always communicates with the active server, without losing any messages.

Oracle BAM Adapter provides three mechanisms by which you can send data to an Oracle BAM Server in your SOA composite application:

- The Oracle BAM Adapter can be used as a reference binding component in an SOA composite application. For example, Oracle Mediator can send data to Oracle BAM using the Oracle BAM Adapter.
- The Oracle BAM Adapter can also be used as a partner link in a Business Process Execution Language (BPEL) process to send data to Oracle BAM as a step in the process.
- Oracle BAM sensor actions can be included within a BPEL process to publish event-based data to the Oracle BAM data objects.

32.2 Configuring Oracle BAM Adapter

The Oracle BAM Adapter Java Naming and Directory Interface (JNDI) connection pools must be configured when you use the adapter (also used with Oracle BAM sensor actions in BPEL) to connect with the Oracle BAM Server at runtime. For information about configuration see "Configuring the Oracle BAM Adapter" in *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite*.

Make note of the JNDI name that you configure in the Oracle BAM Adapter properties, so that you can use them in the Oracle BAM Adapter wizard and the Oracle BAM sensor action configuration in Oracle JDeveloper.

32.3 Creating a Design Time Connection to an Oracle BAM Server

You must create a connection to an Oracle BAM Server to browse the available data objects and construct transformations while you are designing your applications in Oracle JDeveloper.

Notes: Do *not* create an Oracle BAM Server connection through the Resource Palette that is displayed when you select **View > Resource Palette**. The connection must be created in the application scope.

32.3.1 How to Create a Connection to an Oracle BAM Server

You create a connection to an Oracle BAM Server to browse data objects available on that server and to publish data to those data objects.

To create a connection to an Oracle BAM Server:

1. From the **File** main menu in Oracle JDeveloper, select **New**.
The New Gallery dialog box opens.
2. From the **General** category, choose **Connections**.
3. From the **Items** list, select **BAM Connection**, and click **OK**.
The BAM Connection wizard opens.
4. Ensure that **Application Resources** is selected.
5. Provide a name for the connection.
6. Click **Next**.
7. Enter the connection information about the Oracle BAM Server host described in [Table 32-1](#).

Table 32–1 Oracle BAM Server Connection Information

Field	Description
BAM Web Host	Enter the name of the host on which the Oracle BAM Report Server and web applications are installed. In most cases, the Oracle BAM web applications host and Oracle BAM Server host are the same.
BAM Server Host	Enter the name of the host on which the Oracle BAM Server is installed.
User Name	Enter the Oracle BAM Server user name.
Password	Enter the password of the user name.
HTTP Port	Enter the port number or accept the default value of 9001 . This is the HTTP port for the Oracle BAM web applications host.
JNDI Port	Enter the port number or accept the default value of 9001 . The JNDI port is for the Oracle BAM report cache, which is part of the Oracle BAM Server.
Use HTTPS	Select this checkbox to use secure HTTP (HTTPS) to connect to the Oracle BAM Server during design time. Otherwise, HTTP is used.

8. Click **Next**.
9. Test the connection by clicking **Test Connection**. If the connection was successful, the following message appears:

Passed.
10. Click **Finish**.

32.4 Using Oracle BAM Adapter in an SOA Composite Application

The Oracle BAM Adapter is used as a reference that enables the SOA composite application to send data to an Oracle BAM Server external to the SOA composite application.

32.4.1 How to Use Oracle BAM Adapter in an SOA Composite Application

You can add Oracle BAM Adapter references that enable the SOA composite application to send data to Oracle BAM Servers external to the SOA composite application.

To add an Oracle BAM Adapter reference:

1. In the Component Palette, select **SOA**.
2. Drag the **BAM Adapter** to the right swim lane.
This launches the Adapter Configuration wizard.
3. In the Service Name page, provide a **Service Name** and an optional **Description**.
4. In the Data Object Operation and Keys page,
 - a. Select a **Data Object** using the BAM Data Object Chooser dialog box.

When you click **Browse** the Data Object Chooser dialog box opens allowing you to browse the available Oracle BAM Server connections in the **BAM Data Object Explorer** tree. Select a data object and click **OK**.

- b. Choose an **Operation** from the list.

Insert adds a row to the data object.

Upsert inserts new data into an existing row in a data object if the row exists. If the row does not exist a new row is created. You must select a key from the **Available** column to upsert rows in a data object.

Delete removes a row from the data object. You must select a key from the **Available** column to delete rows in a data object.

Update inserts new data into an existing row in a data object. You must select a key from the **Available** column to update rows in a data object.

- c. Provide an appropriate display name in the **Operation Name** field for this operation in your SOA composite application.

- d. To select **Enable Batching** select the checkbox.

The data cached in memory by the Oracle BAM Adapter of the Oracle BPEL Process Manager runtime is flushed (sent) to Oracle BAM Server periodically. The Oracle BAM component may decide to send data before a batch timeout if the cache has some data objects between automatically defined lower and upper limit values.

Batching properties are configured in `BAMCommonConfig.xml`. See *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite* for more information.

5. In the JNDI Name page, specify the **JNDI Name** for the Oracle BAM Server connection.

The JNDI name is configured in the Oracle WebLogic Server Administration Console. See "Configuring the Oracle BAM Adapter" in *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite* for more information.

6. Click **Finish**.

32.5 Using Oracle BAM Adapter in a BPEL Process

The Oracle BAM Adapter is used as a partner link in a BPEL process to send data to Oracle BAM as a step in the process.

See [Section 5.3, "Introduction to Partner Links."](#) for more information.

32.5.1 How to Use Oracle BAM Adapter in a BPEL Process

You can add the Oracle BAM Adapter to a BPEL process to send data to Oracle BAM as a step in the process. The Oracle BAM Adapter is used as a partner link and connected to an activity in the BPEL process.

To add an Oracle BAM partner link:

1. In the SOA Composite Editor in Oracle JDeveloper, double-click the BPEL process icon to open it in the BPEL Process Designer.
2. In the Component Palette, expand the BPEL Services panel.
3. Drag and drop the Oracle BAM Adapter into the Partner Links swim lane on the right side of the BPEL Process Designer.
4. In the Adapter Configuration wizard, enter a display name in the Service Name field and click Next.

When the wizard completes, a Web Services Description Language (WSDL) file by this name appears in the Application Navigator for the BPEL process or Oracle Mediator message flow. This file includes the adapter configuration settings you specify with this wizard.

5. In the Data Object Operation and Keys page,
 - a. Select a **Data Object** using the BAM Data Object Chooser dialog box.

When you click Browse the Data Object Chooser dialog box opens allowing you to browse the available Oracle BAM Server connections in the **BAM Data Object Explorer** tree. Select a data object and click **OK**.
 - b. Choose an **Operation** from the list.

Insert adds a row to the data object.

Upsert inserts new data into an existing row in a data object if the row exists. If the row does not exist a new row is created.

Delete removes a row from the data object.

Update inserts new data into an existing row in a data object.
 - c. Provide an appropriate display name in the **Operation Name** field for this operation in your SOA composite application.
 - d. To select **Enable Batching** select the checkbox.

The data cached in memory by the Oracle BAM Adapter of the Oracle BPEL Process Manager runtime is flushed (sent) to Oracle BAM Server periodically. The Oracle BAM component may decide to send data before a batch timeout if the cache has some data objects between automatically defined lower and upper limit values.

Batching properties are configured in BAMCommonConfig.xml. See *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite* for more information.
6. In the JNDI Name page, specify the **JNDI Name** for the Oracle BAM Server connection.

The JNDI name is configured in the Oracle WebLogic Server Administration Console. See "Configuring the Oracle BAM Adapter" in *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite* for more information.
7. Click **Finish**.
8. Create a new Process Variable in the BPEL process of type Message Type, and browse the Type Chooser dialog box to select the WDSL for the data object you want to write to on the Oracle BAM Server.

For more information about using the Oracle BPEL Process Manager see [Chapter 5, "Getting Started with Oracle BPEL Process Manager."](#)
9. In the BPEL Process add an activity that you can use to map the source data to the new variable you created.
10. In the BPEL Process add an Invoke activity to send data to the Oracle BAM Adapter partner link you created. Add the variable you just created as the Input Variable.
11. Save all of the project files.

32.6 Integrating BPEL Sensors with Oracle BAM

You can create sensor actions in Oracle BPEL Process Manager to publish sensor data into existing data objects on an Oracle BAM Server. When you create the sensor action, you can select an Oracle BPEL Process Manager variable sensor or activity sensor to get the data from and the data object in Oracle BAM Server in which you want to publish the sensor data.

The Oracle BAM Adapter supports batching of operations, but behavior with batching is different from behavior without batching. As the Oracle BAM Adapter is applied to BPEL sensor actions, the Oracle BAM sensor action is not part of the BPEL transaction. When batching is enabled, BPEL does not wait for an Oracle BAM operation to complete. It is an asynchronous call.

When batching is disabled, BPEL waits for the Oracle BAM operation to complete before proceeding with the BPEL process, but it does not roll back or stop when there is an exception from Oracle BAM. The Oracle BAM sensor action logs messages to the same sensor action logger as BPEL. See "Configuring Oracle BAM Batching Properties" in *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite* for information about batching behavior.

These instructions assume you have installed and configured Oracle BAM.

Notes: Connection factory configuration must be completed before using Oracle BAM sensor actions. Also, if the Oracle BAM Adapter is using credentials rather than a plain text user name and password, in order for the Oracle BAM Adapter (including Oracle BAM sensor actions used in BPEL) to connect to the Oracle BAM Server the credentials must also be established and mapped. See "Configuring the Oracle BAM Adapter" in *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite* for more information.

32.6.1 How to Create a Sensor

Before you can create an Oracle BAM sensor action, you must first create a sensor in the BPEL process. You must create a sensor before creating a Oracle BAM sensor action.

- Variable sensor

Restrictions: A Variable sensor's variable must be defined in a standalone XSD. This variable must not be defined inline in the WSDL file. If the variable has message parts, then there must be only one message part.

- An Activity sensor containing exactly one sensor variable

Restrictions: Because you map the sensor data to a single Oracle BAM Server data object, the Activity sensor must contain only one variable. All of the Variable sensor restrictions also apply.

Note: Any sensor that does not conform to these rules are be filtered from the Oracle BAM sensor action configuration dialog box. Also, if a sensor is created conforming to the restrictions, but the variable is deleted (rendering the sensor invalid), it does not appear in Oracle BAM sensor action configuration dialog box.

For more information about creating sensors, see [Section 17.2, "Configuring Sensors and Sensor Actions in Oracle JDeveloper."](#)

32.6.2 How to Create an Oracle BAM Sensor Action

When you create the Oracle BAM sensor action, you select the BPEL variable sensor or activity sensor from which to get data, and you select the data object in Oracle BAM Server to which you want to publish the sensor data.

To create an Oracle BAM sensor action:

1. Go to your BPEL process in Oracle JDeveloper.
2. In the Structure window, right-click **Sensor Actions**.

If the Structure window is not open, select **View > Structure Window** to open it.

3. Select **Create > BAM Sensor Action**.

The Create Sensor Action dialog box appears.

Figure 32–1 Oracle BAM Sensor Action Creation Dialog Box

4. Enter the details described in [Table 32–2](#):

Table 32–2 Create Sensor Action Dialog Box Fields and Values

Field	Description
Action Name	Enter a unique and recognizable name for the sensor action.
Sensor	Select a BPEL sensor to monitor. This is the sensor that you created in Section 32.6.1, "How to Create a Sensor" for mapping sensor data to a data object in Oracle BAM Server.

Table 32–2 (Cont.) Create Sensor Action Dialog Box Fields and Values

Field	Description
Data Object	<p>Click the Browse icon to open the BAM Data Object Chooser dialog box to select the data object in Oracle BAM Server in which you want to publish the sensor data.</p> <p>If you have not created a connection to Oracle BAM Server to select data objects, click the icon in the upper right corner of the BAM Data Object Chooser dialog box.</p>
Operation	Select to Delete , Update , Insert , or Upsert a row in the Oracle BAM Server database. Upsert first attempts to update a row if it exists. If the row does not exist, it is inserted.
Available Keys/Selected Keys	If you selected the Delete , Update , or Upsert operation, you must also select a column name in the Oracle BAM Server database to use as a key to determine the row with which this sensor object corresponds. A key can be a single column or a composite key consisting of multiple columns. Select a key and click the > button. To select all, click the >> button.
Map File	Provide a file name to create a mapping between the sensor data (selected in the Sensor list) and the Oracle BAM Server data object (selected in the Data Object list). You can also invoke a mapper dialog box by clicking the Create Mapping icon (second icon) or Edit Mapping icon (third icon).
BAM Connection Factory JNDI	<p>Specify the JNDI name for the Oracle BAM Server connection factory.</p> <p>The JNDI name is configured in the Oracle WebLogic Server Administration Console. See "Configuring the Oracle BAM Adapter" in <i>Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite</i> for more information.</p>
Enable Batching	<p>The data accumulated by the Oracle BAM component of the Oracle BPEL Process Manager runtime is flushed (sent) to Oracle BAM Server periodically. The Oracle BAM component may decide to send data before a batch timeout if the queue has some data objects between automatically defined lower and upper limit values.</p> <p>If batching is enabled, performance is dramatically improved, but there is no transaction guarantee. The BPEL process continues to run without waiting for the data to get to the Oracle BAM Server.</p> <p>If batching is not enabled, the BPEL process waits until the Oracle BAM Server confirms that the record operation was completed; however, if there is a failure, the exception from Oracle BAM Server is logged and the BPEL process continues. BPEL does not roll back the operation or stop when there is an exception from Oracle BAM.</p> <p>See "Configuring Oracle BAM Batching Properties" in <i>Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite</i> for information about batching behavior.</p>

Figure 32–1 shows the Create Sensor Action dialog box with a selected data object.

WARNING: If you restart Oracle BPEL Server, any messages currently being batched are lost. Ensure that all messages have completed batching before restarting Oracle BPEL Server.

Notes: After you click the **Create Mapping** or **Edit Mapping**, or the **OK** button on the Create Sensor Action dialog box, you must explicitly save the BPEL file.

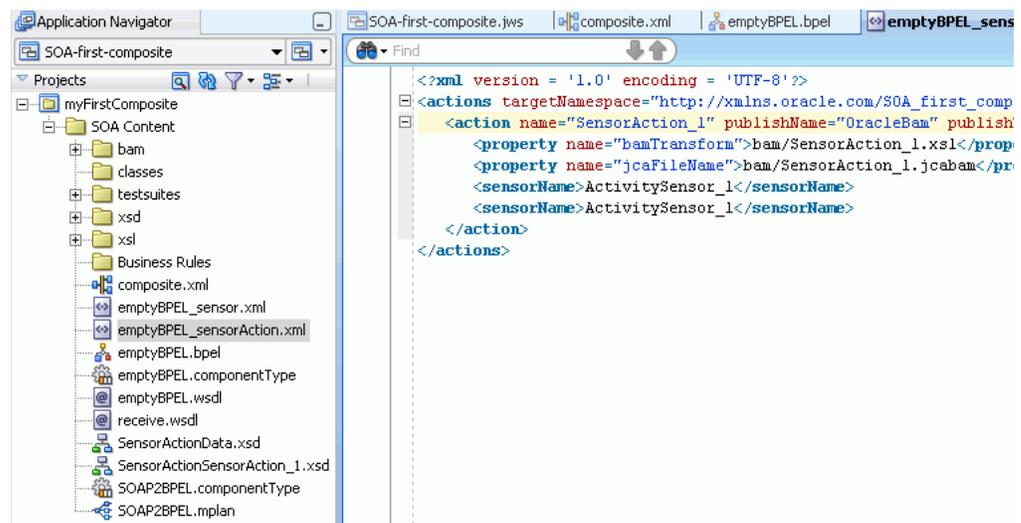
5. Click **OK** to close the Create Sensor Action dialog box.

32.6.3 How to Disable an Oracle BAM Sensor Action

BPEL sensor actions are typically disabled using the Oracle Enterprise Manager Fusion Middleware Control Console; however, Oracle BAM sensor actions are not manageable from the Fusion Middleware Control Console.

To disable an Oracle BAM sensor action:

1. Open the project containing the Oracle BAM sensor action in Oracle JDeveloper.
2. Open the `BPEL_PROCESS_NAME_sensorAction.xml` file in the editor (refresh the project tree if you cannot see the file).



3. Locate the sensor action you want to disable. You can search on the `publishType="BAM"` attribute to find only the Oracle BAM sensor actions.
4. In the `<action>` element for the sensor action you want to disable, set the `enabled` attribute to `false`.



5. After the file is updated and saved, you must redeploy the application.

Creating Oracle BAM Enterprise Message Sources

This chapter contains the information required to create Enterprise Message Sources (EMS) in the Oracle BAM Architect application.

This chapter contains the following topics:

- [Section 33.1, "Introduction to Enterprise Message Sources"](#)
- [Section 33.2, "Creating Enterprise Message Sources"](#)
- [Section 33.3, "Using Foreign JMS Providers"](#)
- [Section 33.4, "Use Case: Creating an EMS Against Oracle Streams AQ JMS Provider"](#)

33.1 Introduction to Enterprise Message Sources

Enterprise Message Sources (EMS) are used by applications to provide direct Java Message Service (JMS) connectivity to the Oracle BAM Server. JMS is the standard messaging API for passing data between application components and allowing business integration in heterogeneous and legacy environments.

The EMS does not configure Extract Transform and Load (ETL) scenarios, but rather maps from a message directly to a data object on the Oracle BAM Server; however, you can still use XML Stylesheet Language (XSL) to perform a transformation in between. Each EMS connects to a specific JMS topic or queue, and the information is delivered into a data object in the Oracle BAM Active Data Cache. The Oracle BAM Architect web application is used to configure EMS definitions.

The following JMS providers are supported:

- Messaging for Oracle WebLogic Server
- Non-Oracle certified JMS providers:
 - IBM WebSphere MQ 6.0
 - Tibco JMS
 - Apache ActiveMQ

See [Section 33.3, "Using Foreign JMS Providers"](#) for more information.

The following message types are supported:

- Map message
- Text message with XML payload

The following XML formatting options are supported for Text message transformation:

- Pre-processing
- Message specification
- Column value (Column values can be provided either as elements or attributes in the XML payload.)

To view the existing EMS definitions, select **Enterprise Message Sources** from the Oracle BAM Architect function list.

Figure 33–1 Oracle BAM Architect Function List



33.2 Creating Enterprise Message Sources

When you define an EMS, you specify all of the fields in the messages to be received. Some messaging systems have a variable number of user-defined fields, while other systems have a fixed number of fields.

For any string type field, you can apply formatting to that field to break apart the contents of the field into separate, individual fields. This is useful for messaging systems where you cannot create user-defined fields and the entire message body is received as one large field. The formatting specifications allow you to specify the path to a location in the XML tree, and then extract the attributes or tags as fields.

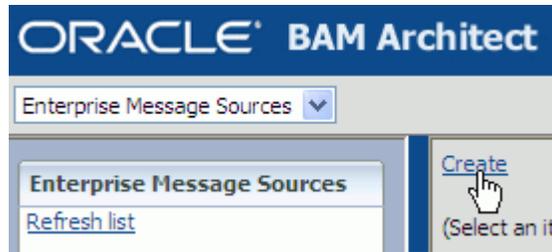
Before defining an EMS, you must be familiar with the third party application providing the messages so that you can specify the message source connection details in Oracle BAM Architect.

Furthermore, note that the JMS server (where you host queues/topics) can be configured on a different system than that which hosts the Oracle BAM Server. (For Oracle Advanced Queuing (AQ) it is acceptable to host on the same server as Oracle BAM because the database hosts the JMS server, but for other cases it is recommended to host the JMS server on another system).

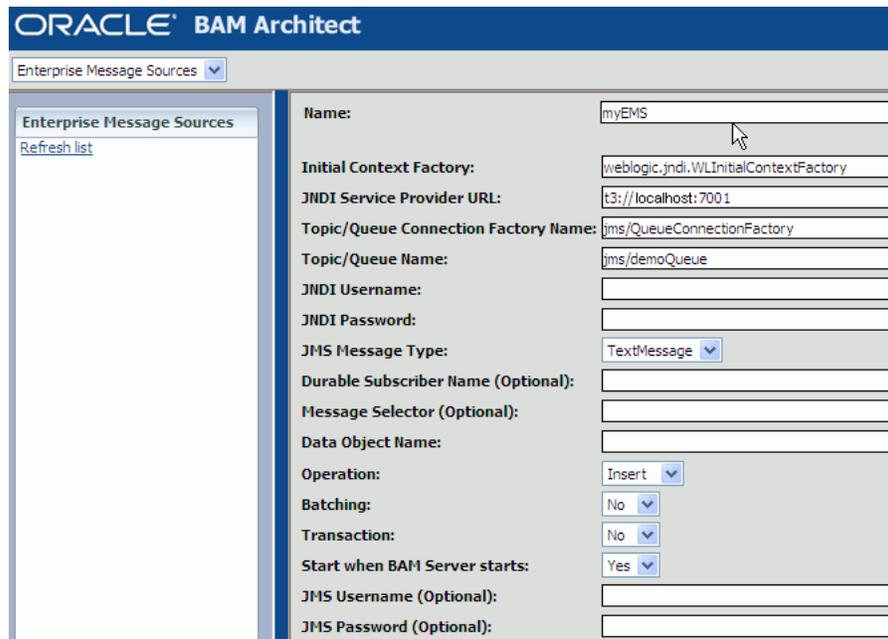
33.2.1 How to Create an Enterprise Message Source

To define an EMS:

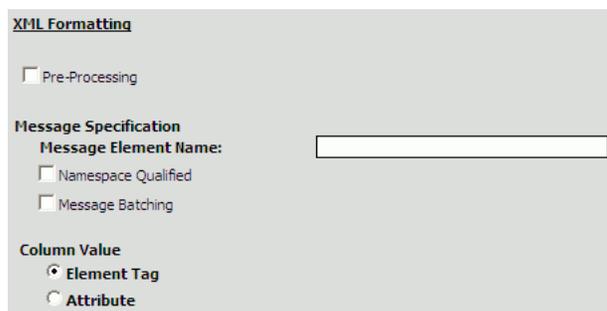
1. Select **Enterprise Message Sources** from the Oracle BAM Architect function list (see [Figure 33–1](#)).
2. Click **Create**.



- Using [Table 33-1](#) as a guide, enter the appropriate values in each of the fields. Examples given are for connecting to Messaging for Oracle WebLogic Server.



- If you are using `TextMessage` type, configure the appropriate parameters in the XML Formatting sections, using [Table 33-2](#) as a guide.



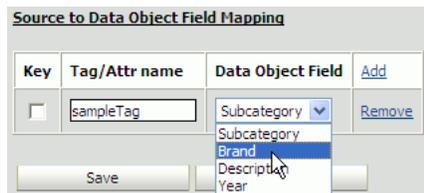
- To configure the **Date Time Specification** in the **Source Value Formatting** section, see [Section 33.2.2, "How to Configure Date Time Specification."](#)

Note that when **Date Time Specification** is disabled (not checked), the incoming value must be in `xsd:dateFormat`. That is, `xsd:dateFormat` (`([-]CCYY-MM-DDThh:mm:ss[Z|(+|-)hh:mm])`) is the default format when **Date Time Specification** is not configured.

Valid value patterns for `xsd:dateTime` include:

- 2001-10-26T21:32:52
- 2001-10-26T21:32:52+02:00
- 2001-10-26T19:32:52Z
- 2001-10-26T19:32:52+00:00
- -2001-10-26T21:32:52
- 2001-10-26T21:32:52.12679

6. Map fields from the source message to the selected data object in the **Source to Data Object Field Mapping** section.



- a. Click **Add** to add a mapped field.
 - b. Select the **Key** checkbox if the field is a key.
 - c. Enter the source tag or attribute name in the **Tag/Attr name** field.
 - d. Select the target data object field from the **Data Object Field** list.
7. Click **Save** to save the EMS.

Table 33–1 EMS Configuration Parameters

Parameter	Description
Name	A unique display name that appears in the EMS list in Oracle BAM Architect.
Initial Context Factory	The initial context factory to be used for looking up specified JMS connection factory or destination. For example: weblogic.jndi.WLInitialContextFactory
JNDI Service Provider URL	Configuration information for the service provider to use. Used to set <code>javax.naming.Context.PROVIDER_URL</code> property and passed as an argument to <code>initialContext()</code> . An incorrect provider URL is the most common cause of errors. For example: t3://localhost:7001
Topic/Queue ConnectionFactory Name	The name used in a JNDI lookup of a previously created JMS connection factory. For example: jms/QueueConnectionFactory
Topic/Queue Name	The name used in the JNDI lookup of a previously created JMS topic or queue. For example: jms/demoQueue jms/demoTopic
JNDI Username	The identity of the principal for authenticating the JNDI service caller. This user must have RMI login permissions. Used to set <code>javax.naming.Context.SECURITY_PRINCIPAL</code> and passed to <code>initialContext()</code> .

Table 33–1 (Cont.) EMS Configuration Parameters

Parameter	Description
JNDI Password	The identity of the principal for authenticating the JNDI service caller. Used to set <code>javax.naming.Context.SECURITY_CREDENTIALS</code> and passed to <code>initialContext()</code> .
JMS Message Type	TextMessage or MapMessage. If TextMessage is selected, XML is used to specify the contents of the payload, and an additional set of XML Formatting configuration parameters must be completed. See Table 33–2 for more information.
Durable Subscriber Name	Enter the name of the subscriber, for example, BAMFilteredSubscription. The Durable Subscriber Name should match the event-publisher subscriber name property if it is provided. A durable subscription can be used to preserve messages published on a topic while the subscriber is not active. It enables Oracle BAM to be disconnected from the JMS provider for periods of time, and then reconnect to the provider and process messages that were published during the disconnected period.
Message Selector (Optional)	A single name-value pair (currently only one name-value pair is supported) that allows an application to have a JMS provider select, or filter, messages on its behalf using application-specific criteria. When this parameter is set, the application-defined message property value must match the specified criteria for it to receive messages. To set message property values, use <code>stringProperty()</code> method on the Message interface.
Data Object Name	Data object in Oracle BAM in which to deposit message data. Operations can be performed on only one data object per EMS. The data object can have Lookup columns. Click Browse to choose a data object.
Operation	Select the operation from the list: Insert inserts all new data as new rows Upsert merges data into existing rows Update updates existing rows Delete removes rows from the data object
Batching	Specify whether the EMS communicates with the Oracle BAM Active Data Cache API with batching enabled. Batching allows multiple messages to be inserted using a single Text Message. If Batching is disabled (the default state), each row read from JMS would be sent to the Active Data Cache as a separate unit and not part of a batch of rows. Batching properties are contained in configuration files. See <i>Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite</i> for more information.
Transaction	Enabling Transaction ensures that the operation is atomic when Batching is enabled (Batching allows multiple messages to be inserted using a single Text Message). Transaction itself does not have any impact on Active Data Cache batching, but setting Transaction to true ensures that all of the messages in Messaging Batching (when many messages are batched in a single batch) are part of an atomic operation. See Message Batching in Table 33–2 .

Table 33–1 (Cont.) EMS Configuration Parameters

Parameter	Description
Start when BAM Server starts	Specify whether the EMS starts reading messages and sending them to the Active Data Cache as soon as the Oracle BAM Server starts (or restarts).
JMS Username (Optional) JMS Password (Optional)	You can optionally provide this information when a new JMS connection is created by a connection factory. Used to authenticate a connection to a JMS provider for either application-managed or container-managed authentication.

Table 33–2 EMS XML Formatting Configuration Parameters

Parameter	Description
Pre-Processing	XSL transformation can be applied to an incoming Text Message before message retrieval and column mapping are done. See Section 33.2.3, "How to Use Advanced XML Formatting" for more information. XML names can be qualified. If qualified, select the Namespace Qualified box and enter the namespace URI in the field.
Message Element Name	The parent element that contains column values in either its sub-elements or attributes. XML names can be qualified. If qualified, select the Namespace Qualified box and enter the namespace URI in the field.
Message Batching	Multiple messages can be batched in a single JMS message. If this is the case, a wrapper element must be specified as the containing element in Batch Element Name . If qualified, select the Namespace Qualified box and enter the namespace URI in the field.
Column Value	Column values can be provided using either elements or attributes in an XML payload. Specify which column value type is provided in the payload.

33.2.2 How to Configure DateTime Specification

To configure DateTime Specification:

1. Select the DateTime Specification checkbox as shown in [Figure 33–2](#).
2. Enter the date and time pattern in the **Pattern** field.

You must supply a valid date and time pattern that adheres to the Java SimpleDateFormat. [Table 33–3](#) provides the syntax elements for SimpleDateFormat, and [Table 33–4](#) provides some examples.

3. Optionally, you can enter the locale information in the **Language**, **Country**, and **Variant** fields.

Figure 33–2 EMS Configuration Source Value Formatting Section

Table 33–3 Syntax Elements for SimpleDateFormat

Symbol	Meaning	Presentation	Example
G	Era	Text	AD
y	Year	Number	2003
M	Month	Text or Number	July; Jul; 07
w	Week in year (1-53)	Number	27
W	Week in month (1-5)	Number	2
D	Day in year (1-365 or 1-364)	Number	189
d	Day in a month	Number	10
F	Day of week in month (1-5)	Number	2
E	Day in week	Text	Tuesday; Tue
a	AM/PM marker	Text	AM
H	Hour (0-23)	Number	0
k	Hour (1-24)	Number	24
K	Hour (0-11 AM/PM)	Number	0
h	Hour (1-12 AM/PM)	Number	12
m	Minute in an hour	Number	30
s	Second in a minute	Number	55
S	Millisecond (0-999)	Number	978
z	Time zone	General time zone	Pacific Standard Time; PST; GMT-08:00
Z	Time zone	RFC 822 time zone	-0800
'	Escape for text	Delimiter	MMM '01 -> Jul '01

The examples in [Table 33–4](#) show how date and time patterns are interpreted in the United States locale. The date and time used in all of the examples are 2001-07-04 12:08:56 local time in the U.S. Pacific Time time zone.

Table 33–4 Date and Time Pattern Examples

Date and Time Pattern	Result
"yyyy.MM.dd G 'at' HH:mm:ss z"	2001.07.04 AD at 12:08:56 PDT
"EEE, MMM d, 'yy"	Wed, Jul 4, '01
"h:mm a"	12:08 PM

Table 33–4 (Cont.) Date and Time Pattern Examples

Date and Time Pattern	Result
"hh 'o'clock' a, zzzz"	12 o'clock PM, Pacific Daylight Time
"K:mm a, z"	0:08 PM, PDT
"yyyyy.MMMMM.dd GGG hh:mm aaa"	02001.July.04 AD 12:08 PM
"EEE, d MMM yyyy HH:mm:ss Z"	Wed, 4 Jul 2001 12:08:56 -0700
"yyMMddHHmmssZ"	010704120856-0700
"yyyy-MM-dd'T'HH:mm:ss.SSSZ"	2001-07-04T12:08:56.235-0700

33.2.3 How to Use Advanced XML Formatting

The Advanced formatting options allow an EMS to contain a user-supplied XSL Transformation (XSLT) for each formatted field in the message.

Uses for XSLT include:

- Handling of hierarchical data. The Data Flow does not handle hierarchical data. The XSLT can flatten the received XML into a single record with repeating fields.
- Handling of message queues that contain messages of multiple types in a single queue. The Data Flow requires that all records from the Message Receiver be of the same schema. The EMS output can be defined as a combined superset of the message schemas that are received, and the XSL transformation can identify each message type and map it to the superset schema as appropriate.
- Handling of XML that, while not expressing hierarchical data, does contain needed data at multiple levels in the XML. EMS formatting can only read from one level with the XML. The XSL transformation can identify the data needed at various levels in the input XML and output it all in new XML that contains all of the data combined at one level.

To specify an XSL transformation:

1. In an EMS that you are defining or editing, select **Pre-Processing** in the XML Formatting section.



2. Click **Advanced formatting options**.
The Advanced Formatting dialog box opens.
3. Type or paste the XSL markup for the transformation for the XML in this field. You might want to write the XSL markup in another editing tool and then copy and paste the code into this dialog box.
4. In the **Sample XML to transform** field, type sample XML to test the transformation against. The sample XML is not saved in this dialog box and is not be displayed if you close and open this dialog box.
5. Click **Verify transformation syntax** to validate the XSL syntax.

6. Click **Test transformation on sample XML** to test your transformation.

The results are displayed in the field underneath the links. If any errors are found in the XSL syntax, the sample XML syntax, or during the transformation, the error text is shown in this field.

33.3 Using Foreign JMS Providers

Oracle WebLogic Server provides support for integrating non-Oracle WebLogic Server (foreign) JMS providers with applications deployed in it, such as Oracle BAM. Foreign JMS providers have their own JMS client and Java Naming and Directory Interface (JNDI) Client APIs. Some configuration must be done to identify these dependencies and make these APIs available on Oracle WebLogic Server so that JMS resources hosted on a remote provider can be looked up by application deployed in Oracle WebLogic Server.

See "Configuring Foreign Server Resources to Access Third-Party JMS Providers" in *Oracle Fusion Middleware Configuring and Managing JMS for Oracle WebLogic Server* for more information.

[Section 33.4.3, "Creating a Foreign JMS Server"](#) in the "Use Case: Creating an EMS Against Oracle Streams AQ JMS Provider" provides a detailed example.

The high level configuration steps are:

1. Make the JMS and JNDI client library of the foreign the provider available to applications deployed on Oracle WebLogic Server.

Identify the JMS and JNDI client Java Archive (JAR) files of the foreign provider and place them in the `DOMAIN_HOME/lib` directory.

2. Create a foreign server using Oracle WebLogic Server Administration Console.

Go to **JMS Modules** in Oracle WebLogic Server Administration Console, and create a new module.

Inside this module, click **New**, select **Foreign Server**, and create a new foreign server by navigating through all of the pages.

Provide appropriate JNDI properties for the remote provider for the foreign server definition.

3. Create JMS resources (that is, connection factories and destinations) for the foreign JMS server.

Inside the **Foreign Server** link, select the **Destination** tab and create links for

- Remote ConnectionFactory
- Remote Destination (Queue/Topic)

Local JNDI names configured for these destinations must be used while configuring EMS to consume messages from these destinations.

4. Configure an EMS definition in Oracle BAM Architect to consume messages from foreign destinations.

The whole process of accessing JMS resources hosted on foreign providers is transparent to Oracle BAM Server. After the previous steps have been followed correctly, remote destinations from foreign JMS providers are published on the local WL server JNDI tree, so that applications deployed on the server (like Oracle BAM EMS) can look them up, just like any other collocated Oracle WebLogic

Server JMS resource. Oracle WebLogic Server takes care of communicating with the appropriate foreign JMS provider at runtime.

33.4 Use Case: Creating an EMS Against Oracle Streams AQ JMS Provider

The following are the steps to configure Oracle Streams AQ JMS Provider (AQ-JMS) in Oracle WebLogic Server, and an EMS definition in Oracle BAM Architect.

1. [Creating a JMS Topic in AQ-JMS.](#)
2. [Creating a Data Source in Oracle WebLogic Server.](#)
3. [Creating a Foreign JMS Server.](#)
4. [Defining an EMS in Oracle BAM Architect.](#)
5. [Inserting and Updating Records in the SQL Table.](#)

33.4.1 Creating a JMS Topic in AQ-JMS

Open a SQLplus command prompt and do the following:

1. Login as sysdba


```
sqlplus sys as sysdba
```
2. Enter the password for the system dba account when prompted.
3. Create and execute the following scripts in the following order (see [Example 33-1](#), [Example 33-2](#), and [Example 33-3](#) for the contents of the scripts).

```
@<SCRIPT_PATH>/usertabletopiccreation.sql
@<SCRIPT_PATH>/createtable.sql
@<SCRIPT_PATH>/createtrigger.sql
```

The scripts do the following things:

- a. Creates a fresh schema under user MyChannelDemoUser.
- b. Creates a JMS a topic in AQ-JMS.
- c. Creates a SQL table by name EMP.
- d. Creates a trigger that publishes messages to AQ-JMS topic on insert/update on EMP.

Example 33-1 Contents of usertabletopiccreation.sql

```
DROP USER MyChannelDemoUser CASCADE;

GRANT connect, resource,AQ_ADMINISTRATOR_ROLE TO MyChannelDemoUser IDENTIFIED BY
  MyChannelDemoPassword;
GRANT execute ON sys.dbms_aqadm TO MyChannelDemoUser;
GRANT execute ON sys.dbms_aq TO MyChannelDemoUser;
GRANT execute ON sys.dbms_aqin TO MyChannelDemoUser;
GRANT execute ON sys.dbms_aqjms TO MyChannelDemoUser;

connect MyChannelDemoUser/MyChannelDemoPassword;

BEGIN
--dbms_aqadm.stop_queue( queue_name => 'MY_TOPIC' );
--dbms_aqadm.drop_queue( queue_name => 'MY_TOPIC');
```

```
--DBMS_AQADM.DROP_QUEUE_TABLE (queue_table => 'TTab');
dbms_aqadm.create_queue_table( queue_table => 'TTab', queue_payload_type =>
'sys.aq$_jms_text_message', multiple_consumers => true );
dbms_aqadm.create_queue( queue_name => 'MY_TOPIC', queue_table => 'TTab' );
dbms_aqadm.start_queue( queue_name => 'MY_TOPIC' );
END;
/
```

Example 33–2 Contents of createtable.sql

```
connect MyChannelDemoUser/MyChannelDemoPassword;

CREATE TABLE EMP ( EMPNO NUMBER(4), ENAME VARCHAR2(10), JOB VARCHAR2(9), MGR
NUMBER(4), HIREDATE DATE, SAL NUMBER(7,2), COMM NUMBER(7,2), DEPTNO NUMBER(2) );

quit;
```

Example 33–3 Contents of createtrigger.sql

```
connect MyChannelDemoUser/MyChannelDemoPassword;
create or replace
trigger employee AFTER INSERT OR Update ON EMP
FOR each row
declare
    xml_complete varchar2(1000);
    v_enqueue_options dbms_aq.enqueue_options_t;
    v_message_properties dbms_aq.message_properties_t;
    v_msgid raw(16);
    temp sys.aq$_jms_text_message;
    v_recipients dbms_aq.aq$_recipient_list_t;

Begin
    temp:=sys.aq$_jms_text_message.construct;
xml_complete :=
'<?xml version="1.0"?>' ||
'<row>' ||
'<EMPNO>' || :new.EMPNO || '</EMPNO>' ||
'<ENAME>' || :new.ENAME || '</ENAME>' ||
'<JOB>' || :new.JOB || '</JOB>' ||
'<MGR>' || :new.MGR || '</MGR>' ||
'<HIREDATE>' || :new.HIREDATE || '</HIREDATE>' ||
'<SAL>' || :new.SAL || '</SAL>' ||
'<COMM>' || :new.COMM || '</COMM>' ||
'<DEPTNO>' || :new.DEPTNO || '</DEPTNO>' ||
'</row>' ;
temp.set_text(xml_complete);
    dbms_aq.enqueue(queue_name => 'MY_TOPIC',
enqueue_options => v_enqueue_options,
message_properties => v_message_properties,
payload => temp,
msgid => v_msgid );

End ;
/
quit;
```

33.4.2 Creating a Data Source in Oracle WebLogic Server

You can skip this step if a data source exists. An existing data source can also be reused in this section.

1. Open Oracle WebLogic Server Administration Console at
`http://host_name:7001/console`
where *host_name* is the name of the system where Oracle BAM Server is installed.
2. After logging into the console click the **Data Sources** link in the **JDBC** section, and click **New**.
3. Enter a name for the data source (For example, *BAMAQDataSource*).
4. Enter a JNDI name from the data source (for example, *jdbc/oracle/bamaq*). This name is used to configure the foreign JMS server.
5. Select **Oracle** to be the **Database Type**.
6. Select **Oracle's Driver (Thin)** for **Database Driver** field, and click **Next**.
7. Uncheck **Support Global Transaction**, and click **Next**.
8. Enter your database SID in the **Database Name** field (for example, *ORCL*).
9. Enter the hostname of the system where the database is installed as the **Host Name** (for example, *localhost*).
10. Enter data base port number (for example, *1521*).
11. Enter the user name (for example, *MyChannelDemoUser*).
12. Enter the password, and click **Next**.
13. Click *Test Configuration* to test the configuration.
14. After it is successful, click **Finish**.

33.4.3 Creating a Foreign JMS Server

To create a foreign JMS server:

1. Add as an Oracle WebLogic Server JMS module.
 - a. In the Oracle WebLogic Server Administration Console, from the home page, go to the JMS Modules page.
 - b. Click **New** to create an Oracle WebLogic Server JMS module.
 - c. Enter a name for the JMS module (for example, *BAMAQsystemModule*).
 - d. Click **Next** and assign appropriate targets.
 - e. Click **Next**, and click **Finish**.
2. Add an AQ-JMS foreign server to the JMS module.
 - a. Select the JMS module that you just created.
 - b. Click **New**, and go to the list of JMS resources to add.
 - c. Select the **Foreign Server** option, and click **Next**.
 - d. Enter a name for the foreign server (for example, *BAMAQForeignServer*), and click **Finish**.
3. Configure the AQ-JMS foreign server.

- a. Select the AQ-JMS foreign server that you created.
 - b. In the **JNDI Initial Context Factory** field, enter
`oracle.jms.AQjmsInitialContextFactory`
 - c. In the **JNDI Properties** area, enter
`datasource=datasource_jndi_location`
 where `datasource_jndi_location` is the JNDI location of your data source (for example, `jdbc/oracle/bamaq`).
4. Add connection factories to the AQ-JMS foreign server.
 - a. Select the AQ-JMS foreign server that you created.
 - b. Select the **Connection Factories** tab.
 - c. Enter a name for the connection factory. This is a logical name referenced by Oracle WebLogic Server.
 - d. In the **Local JNDI Name** field, enter the local JNDI name that is used by The Oracle BAM EMS to look up this connection factory (For example, `jms/BAMAQTopicCF`).
 - e. In the **Remote JNDI Name** field, enter:
 - `TopicConnectionFactory` (select for this use case)
 - `QueueConnectionFactory`
 - `ConnectionFactory`
 - f. Click **OK**.
 5. Add destinations to the AQ-JMS foreign server.
 - a. Select the AQ-JMS foreign server that you created.
 - b. Select the **Destinations** tab.
 - c. Enter a name for this destination. This is a logical name referenced by Oracle WebLogic Server, and it has nothing to do with the destination name.
 - d. In the **Local JNDI Name** field, enter the local JNDI name that is used by the Oracle BAM EMS to lookup this destination (for example, `jms/BAMAQTopic`).
 - e. In the **Remote JNDI Name** field, if the destination is a queue, enter the following value:
`Queues/queue_name`

 If the destination is a topic enter the following value:
`Topics/topic_name`
 - f. Click **OK**.
 6. Restart Oracle WebLogic Server.

33.4.4 Defining an EMS in Oracle BAM Architect

1. Open Oracle BAM Architect, and select **Enterprise Message Sources** in the dropdown list.
2. Enter the message source information you just created.
3. Enter the **Initial Context Factory** value:

```
weblogic.jndi.WLInitialContextFactory
```

4. Enter the JNDI provider URL:

```
t3://host_name:7001
```

5. Enter the **Connection Factory Name** (for example, `jms/BAMAQTopicCF`).
6. Enter the **Destination Name** (for example, `jms/BAMAQTopic`).
7. Choose the Oracle BAM data object to send the values received from AQ-JMS server.
8. Complete the source-to-data object field mapping so that data from the incoming XML can be mapped to an appropriate field in selected data object.

33.4.5 Inserting and Updating Records in the SQL Table

Now you can test the functionality end to end by inserting or updating some records in the EMP database table.

You can use SQLPlus to run SQL queries.

Now you should see the values from the record being inserted into data object.

For example,

```
insert into emp values (25, 'Ford', 'ANALYST', 7566, sysdate, 60000, 3000, 20);
```

```
update emp set ENAME='McOwen' where ENAME='Ford';
```

Using Oracle Data Integrator With Oracle BAM

This chapter provides information about the Oracle Data Integrator integration with Oracle Business Activity Monitoring. It contains the following topics:

- [Section 34.1, "Introduction to Using the Oracle Data Integrator With Oracle Business Activity Monitoring"](#)
- [Section 34.2, "Installing the Oracle Data Integrator Integration Files"](#)
- [Section 34.3, "Creating the Oracle BAM Target"](#)
- [Section 34.4, "Using Oracle BAM Knowledge Modules"](#)
- [Section 34.5, "Updating the Oracle Data Integrator External Data Source Definition"](#)
- [Section 34.6, "Launching Oracle Data Integrator Scenarios From Oracle BAM Alerts"](#)

Oracle Data Integrator documentation is located on the Oracle Technology Network web site at the following location:

http://www.oracle.com/technology/products/oracle-data-integrator/10.1.3/htdocs/1013_support.html

34.1 Introduction to Using the Oracle Data Integrator With Oracle Business Activity Monitoring

This document assumes the following:

- The Oracle database is installed and you can connect to it.
- Oracle BAM is installed and running.
- Oracle Data Integrator installed and the basic configuration is done (the Oracle Data Integrator Master repository is created, repository connections are configured, Work repositories are created and connected, and any source topologies are configured).
- If Oracle Data Integrator is installed on a separate host, Java 1.6 must be installed on the Oracle Data Integrator host before you can work with the Oracle BAM and Oracle Data Integrator integration.

When using Oracle Data Integrator with Oracle BAM, keep the following in mind:

- Within the Oracle Data Integrator interface you must add quotation marks around field names that contain spaces.
- Oracle Data Integrator cannot insert data into Oracle BAM read-only fields of type Lookup, Calculated, Auto-incrementing integer, and Timestamp. These fields are automatically populated.
- Do not use Oracle BAM as a staging area (for example, if Oracle BAM is used as a source (as when using a loading knowledge module), do not use this source as staging area, and if Oracle BAM is being used as a target (as when using an integration knowledge module) do not use that target as staging area.

34.2 Installing the Oracle Data Integrator Integration Files

There are two ways to set up the Oracle BAM and Oracle Data Integrator integration.

The first method uses an installation script, typically when Oracle Data Integrator and Oracle BAM are deployed on the same system or the same network file system (Section 34.2.1, "How to Install Integration Files Using the Script").

The second method uses manual steps to configure the properties and copy the required files to the Oracle Data Integrator directories (Section 34.2.2, "How to Manually Install Integration Files"). This method is typically used if you are unable to map the `ODI_HOME` drive from the system where Oracle BAM is installed (usually when Oracle Data Integrator and Oracle BAM are installed in different network or file system).

34.2.1 How to Install Integration Files Using the Script

Use the installation script when you have Oracle Data Integrator and Oracle BAM installed on the same system or the same network file system.

To install the integration files:

1. On the Oracle BAM host, go to the `ORACLE_HOME\bam\config` directory and edit the `bam_odi_configuration.properties` file.

- **ODI_HOME**

This property identifies the path to the Oracle Data Integrator home directory.

The default value on Linux is `/scratch/$user/ODI_HOME/oracledi`.

On Microsoft Windows systems, use the short 8-character name convention.

Also, use double back-slashes (`\\`) to denote a directory separator. For example, `C:\Program Files\ODI_HOME\oracledi` would appear as:

```
ODI_HOME = C:\\Progra~1\\ODI_HOME\\oracledi
```

Note: If Oracle BAM Server and Oracle Data Integrator are deployed on two different machines, then you must map the Oracle Data Integrator drive on the Oracle BAM system, and then set the `ODI_HOME` path using that mapped drive to successfully make use of the integration configuration scripts. If drive mapping is not possible see Section 34.2.2, "How to Manually Install Integration Files."

- **WL_SERVER**

This property identifies the Oracle WebLogic Server folder name on the Oracle BAM system.

The default value is `wlserver_10.3`.

2. Execute `bam_odi_configuration.sh` (or `bam_odi_configuration.bat` on a Microsoft Windows host) in `ORACLE_HOME\bam\bin`.

Figure 34–1 Integration Configuration Script User Input

```
ade:                                     ]$ ./bam_odi_configuration.sh
JAVA_HOME set is: /ade/                  /jdk6

create_eds_in_bam:
Please provide BAM credentials:
Input BAM UserName [OracleSystemUser].....:
Input BAM User login password .....:

Creating Enterprise Data Sources in BAM

Please input connection information for ODI repositories
Input db instance host name or IP address [localhost].....:
Input db instance port number [1521].....:
Input db instance ID/SID [ORCL].....:xe

Creating ODI Master External Data Source:
Input Master repository login name [ODI_M].....:
Input Master repository login password.....:
Importing from file "/                  /Oracle_SOA2/bam/config/icommand_temp22949.xml".
External Data Source "ODI_Master" imported successfully.
Items were imported from "1" files.

Creating ODI Work External Data Source:
Input Work repository login name [ODI_W].....:
Input Work repository login password.....:
```

Enter the values as prompted by the script, as shown in [Figure 34–1](#) (using an Oracle XE database as an example). You must have the Oracle Data Integrator Master and Oracle Data Integrator Work repository account credentials to complete the script execution.

Note that the prompts displayed with `[value]` have default values in the brackets. Press **Enter** to choose the default. If there is no bracketed default value displayed, an input value is required, or the script stops.

The script creates the resources required in the Oracle BAM web applications, sets the Oracle BAM configuration properties in Oracle Data Integrator, generates a Oracle WebLogic Server client Java Archive (JAR) to deploy to the Oracle Data Integrator system, and copies all of the required files into the appropriate Oracle Data Integrator directories.

Note: If you cannot use the script in your environment, use the instructions in [Section 34.2.2, "How to Manually Install Integration Files."](#)

3. After running the script, edit the `ODI_HOME/oracledi/lib/config/BAMCommonConfig.xml` file, and update the entries for `ADCServerName` and `ADCServerPort` to the hostname and port number values where Oracle BAM Server is running.

4. Shut down and restart the Oracle Data Integrator Topology Manager and Designer applications to load the changes you made to the `BAMCommonConfig.xml` file.

Now you can create an Oracle BAM target in the Oracle Data Integrator Topology Manager. See [Section 34.3, "Creating the Oracle BAM Target"](#) for instructions.

34.2.2 How to Manually Install Integration Files

Use these steps if Oracle Data Integrator and Oracle BAM Server are installed on machines in different networks, or for any reason you cannot use the script in your environment.

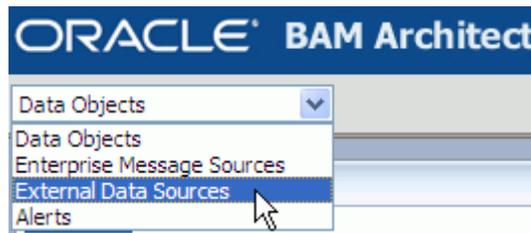
There are four major steps to this process:

1. [Create External Data Sources for Oracle Data Integrator](#)
2. [Set Oracle Data Integrator Configuration Parameters](#)
3. [Copy files to Oracle Data Integrator Directories](#)
4. [Generate the Oracle WebLogic Server Client JAR](#)
5. [Edit the BAMCommonConfig.xml File](#)

Create External Data Sources for Oracle Data Integrator

Create the external data sources in Oracle BAM Architect.

1. Open Oracle BAM Architect and select the External Data Sources page.



2. Click **Create**, and configure the two external data sources (`ODI_Master` and `ODI_Work`) with the values shown in [Table 34–1](#) and [Table 34–2](#).

Table 34–1 ODI_Master external data source values

Property	Value
External Data Source Name	<code>ODI_Master</code>
Driver	<code>oracle.jdbc.driver.OracleDriver</code>
Login	Oracle Data Integrator Master repository account user name
Password	Oracle Data Integrator Master repository account password
Connection String	<code>jdbc:oracle:thin:ip_address:port_number:db_service_name</code>

Table 34–2 ODI_Work external data source values

Property	Value
External Data Source Name	<code>ODI_Work</code>
Driver	<code>oracle.jdbc.driver.OracleDriver</code>
Login	Oracle Data Integrator Work repository account user name

Table 34–2 (Cont.) ODI_Work external data source values

Property	Value
Password	Oracle Data Integrator Work repository account password
Connection String	jdbc:oracle:thin:ip_address:port_number:db_service_name

Set Oracle Data Integrator Configuration Parameters

Modify the `ODI_ADDITIONAL_JAVA_OPTIONS` and `ODI_ADDITIONAL_CLASSPATH` values in the `odiparams.sh` (bat) file located in `ODI_HOME/bin` as shown in [Example 34–1](#) and [Example 34–2](#).

Example 34–1 ODI_ADDITIONAL_JAVA_OPTIONS Modification

```
ODI_ADDITIONAL_JAVA_OPTIONS="-Djava.util.logging.config.file=../lib/bam_odi.logging.properties"
```

Example 34–2 ODI_ADDITIONAL_CLASSPATH Modification

```
ODI_ADDITIONAL_CLASSPATH=../lib/weblogic/wlfullclient.jar
```

Copy files to Oracle Data Integrator Directories

This procedure copies several JAR files, logging properties, and knowledge modules into the Oracle Data Integrator directories.

- Copy the following files from `ORACLE_HOME/bam/modules/oracle.bam_11.1.1` to `ODI_HOME/lib`:
 - oracle-bam-common.jar
 - oracle-bam-etl.jar
 - oracle-bam-adc-ejb.jar
- Copy the following files from `ORACLE_HOME/bam/modules/oracle.bam.thirdparty_11.1.1` to `ODI_HOME/lib`:
 - commons-codec-1.3.jar
 - xstream-1.1.3.jar
- Copy the following file from `ORACLE_HOME/modules/oracle.odl_11.1.1` to `ODI_HOME/lib`:
 - ojdl.jar
- Copy the following file from `ORACLE_HOME/modules/oracle.jps_11.1.1` to `ODI_HOME/lib`:
 - jps-api.jar
- Copy the following file from `ORACLE_HOME/modules/oracle.dms_11.1.1` to `ODI_HOME/lib`:
 - dms.jar

6. Copy the following file from `ORACLE_HOME/modules` to `ODI_HOME/lib`:
 - `org.jaxen_1.1.1.jar`
7. Copy the following file from `ORACLE_HOME/bam/config` to `ODI_HOME/lib`:
 - `bam.odi.logging.properties`
8. Copy the following file from `ORACLE_HOME/bam/ODI/config` to `ODI_HOME/lib/config`:
 - `BAMCommonConfig.xml`
9. Copy all of the XML files from `ORACLE_HOME/bam/odi/knowledge_modules` to `ODI_HOME/impexp`.

Generate the Oracle WebLogic Server Client JAR

1. Generate a `wlfullclient.jar` file using the Oracle WebLogic Server JarBuilder tool. See "Using the WebLogic JARBuilder tool" in *Oracle Fusion Middleware Programming Stand-alone Clients for Oracle WebLogic Server* for instructions.
2. Create a subdirectory called `ODI_HOME/oracledi/lib/weblogic`.
3. Copy `wlfullclient.jar` into `ODI_HOME/oracledi/lib/weblogic`.

Edit the BAMCommonConfig.xml File

1. Edit the `ODI_HOME/oracledi/lib/config/BAMCommonConfig.xml` file, and update the entries for `ADCServerName` and `ADCServerPort` to the hostname and port number values where Oracle BAM Server is running.
2. Shut down and restart the Oracle Data Integrator Topology Manager and Designer applications to load the changes you made to the `BAMCommonConfig.xml` file.

34.3 Creating the Oracle BAM Target

This section details the steps for creating an Oracle BAM target using the Oracle Data Integrator Topology Manager.

For more information about using Oracle Data Integrator, see the Oracle Data Integrator documentation located on the Oracle Technology Network web site at:

http://www.oracle.com/technology/products/oracle-data-integrator/10.1.3/htdocs/1013_support.html

34.3.1 How to Create the Oracle BAM Target

To create an Oracle BAM Target in Oracle Data Integrator:

1. Open the Oracle Data Integrator Topology Manager.
2. Go to **Physical Architecture > Technologies > Oracle BAM**.
3. Right-click and choose **Insert Data Server**.

4. Configure the following in the **Data Server Definition** tab:
 - **Name:** Oracle BAM target name
 - **Server (Data Server):** leave blank
 - **User:** Oracle BAM Administrator user name
 - **Password:** Oracle BAM Administrator password
5. Configure the following in the **JDBC** tab:
 - **JDBC Driver:** any_text_will_do
 - **JDBC URL:** `instance1:host_name:port_number`
 The *host_name* value must be the same as the `ADCServerName` property value in the `BAMCommonConfig.xml` file, and the *port_number* value must be the same as the `ADCServerPort` property value in the `BAMCommonConfig.xml` file.
 - Do not use the **Test** button in this dialog box, because it is not functional for the integration between Oracle BAM and Oracle Data Integrator. After you successfully reverse engineer the data objects in the Oracle BAM model, then you can verify that the connection information is correct.
6. Click **OK**.
7. Configure the following in the Physical Data Server dialog box:
 - In the **Physical Schema Definition** tab:
 - Modify the **Local Object Mask** to be `%OBJECT`.
 - In the **Context** tab:
 - Create a new row which automatically introduces a row with the **Context** name `Global`.
 For that row, the **Logical Schema** value is initially `<Undefined>`. You must select the `<Undefined>` text and replace it with the display name for Oracle BAM.
 - Type in a display name for the Oracle BAM target such as `BAM_TARGET` as the name of a new **Logical Schema**. Oracle Data Integrator automatically creates the logical schema.
 - Click **OK**.

34.4 Using Oracle BAM Knowledge Modules

Knowledge modules are generic code templates containing the sequence of commands necessary for a data integration pattern. A knowledge module contains the knowledge required by Oracle Data Integrator to perform a specific set of tasks against a specific storage technology. It defines methods related to a given storage technology and it enables processes generation for that technology.

There are different knowledge modules for loading (from the source data store), integration (to target data store), checking, reverse-engineering, journalizing and creating services. All knowledge modules work by generating code to be executed at runtime by knowledge module Interpreter.

There is a set of knowledge modules specific to Oracle BAM functionality within Oracle Data Integrator. These knowledge modules are installed in the `ODI_HOME/oracledi/impexp` directory when the integration files are installed. To use

these Oracle BAM-specific knowledge modules, you must import them into the appropriate projects in the Oracle Data Integrator Designer application. [Table 34-3](#) describes these Oracle BAM-specific knowledge modules.

Table 34-3 Oracle BAM Knowledge Modules

Knowledge Module	Description
CKM Get Oracle BAM Metadata	<p>A check knowledge module that is used internally before integration knowledge module steps. This check knowledge module is the default knowledge module in Oracle BAM technology, and it is automatically acquired by Oracle Data Integrator. This check knowledge module creates two arrays which are later used by Oracle BAM-specific integration knowledge modules in the same Java session.</p> <p>This knowledge module has no options.</p>
IKM SQL to Oracle BAM (delete)	<p>An integration knowledge module that can be used to delete rows from Oracle BAM data objects by sending matching key column values. It has the following options:</p> <p>COMMIT_SIZE BATCH_SIZE DATETIME_PATTERN KEY_CONDITION LAST_BAM_TASK LOCALE_COUNTRY LOCALE_LANGUAGE LOCALE_VARIANT</p>
IKM SQL to Oracle BAM (insert)	<p>An integration knowledge module that can be used to insert rows to Oracle BAM data objects from heterogeneous data sources. It has the following options:</p> <p>BATCH_SIZE COMMIT_SIZE CREATE_TARG_TABLE DATETIME_PATTERN LAST_BAM_TASK LOCALE_COUNTRY LOCALE_LANGUAGE LOCALE_VARIANT</p>

Table 34-3 (Cont.) Oracle BAM Knowledge Modules

Knowledge Module	Description
IKM SQL to Oracle BAM (looksert natural)	<p data-bbox="808 260 1445 420">An integration knowledge module that can be used to insert rows into Oracle BAM data objects from heterogeneous data sources. It differs from IKM SQL to Oracle BAM (insert) by also inserting new entries in dimension tables (that is, the data object to which the lookup column refers) if it does not yet exist.</p> <p data-bbox="808 436 1445 646">Looksert integration knowledge modules do an insert into an Oracle BAM target based on a lookup field. Typically, this is used to load a fact table in a star schema. (A star schema is characterized by one or more very large fact tables that contain the primary information in the data warehouse, and some much smaller dimension tables (or lookup tables), each of which contains information about the entries for a particular attribute in the fact table.)</p> <p data-bbox="808 663 1445 709">This integration knowledge module is provided for better performance. It has the following options:</p> <p data-bbox="808 726 1078 1024"> BATCH_SIZE COMMIT_SIZE DATETIME_PATTERN LAST_BAM_TASK LOCALE_COUNTRY LOCALE_LANGUAGE LOCALE_VARIANT NON_KEY_MATCHING </p>
IKM SQL to Oracle BAM (looksert surrogate)	<p data-bbox="808 1050 1445 1176">An integration knowledge module that can be used to insert rows into Oracle BAM data objects from heterogeneous data sources. It is similar to IKM SQL to Oracle BAM (looksert natural) and differs in using a surrogate key instead of a natural key between a fact data object and dimension object.</p> <p data-bbox="808 1192 1445 1402">Looksert integration knowledge modules do an insert into an Oracle BAM data object based on a lookup field. Typically, this used to load a fact table in a star schema. (A star schema is characterized by one or more very large fact tables that contain the primary information in the data warehouse, and some much smaller dimension tables (or lookup tables), each of which contains information about the entries for a particular attribute in the fact table.)</p> <p data-bbox="808 1419 1445 1465">If the value for a lookup field does not exist in the relevant dimension table, the value is automatically inserted.</p> <p data-bbox="808 1482 1445 1562">This integration knowledge module must be used with LKM Get Source Metadata and CKM Get Oracle BAM Metadata.</p> <p data-bbox="808 1579 1445 1604">This knowledge module has the following options:</p> <p data-bbox="808 1621 1078 1923"> BATCH_SIZE COMMIT_SIZE DATETIME_PATTERN LAST_BAM_TASK LOCALE_COUNTRY LOCALE_LANGUAGE LOCALE_VARIANT NON_KEY_MATCHING </p>

Table 34–3 (Cont.) Oracle BAM Knowledge Modules

Knowledge Module	Description
IKM SQL to Oracle BAM (update)	<p>An integration knowledge module that can be used to update rows in Oracle BAM data objects from heterogeneous data sources. It has the following options:</p> <p>BATCH_SIZE COMMIT_SIZE DATETIME_PATTERN LAST_BAM_TASK LOCALE_COUNTRY LOCALE_LANGUAGE LOCALE_VARIANT</p>
IKM SQL to Oracle BAM (upsert)	<p>An integration knowledge module that can be used to merge (upsert) rows (that is, update a data object if matching row exists or insert data object if a new row) to Oracle BAM data objects from heterogeneous data sources. It has the following options:</p> <p>BATCH_SIZE COMMIT_SIZE DATETIME_PATTERN LAST_BAM_TASK LOCALE_COUNTRY LOCALE_LANGUAGE LOCALE_VARIANT</p> <p>Note: During execution, the number of upsert operations are reported in the No. of Updates field, because the Oracle Data Integrator Operator user interface does not have a No. of Upserts field.</p>
LKM Get Source Metadata	<p>A loading knowledge module. This is not a traditional loading knowledge module because it does not load any data from the source to staging area. Instead it simply gathers the metadata that is required by the integration knowledge module IKM SQL to Oracle BAM (looksert surrogate).</p> <p>IKM ORACLE to BAM (looksert surrogate) performs the task of loading directly from a SQL source into the Oracle BAM target. In doing so, it uses the metadata provided by LKM Get Source Metadata.</p> <p>This knowledge module has no options.</p>

Table 34–3 (Cont.) Oracle BAM Knowledge Modules

Knowledge Module	Description
LKM Oracle BAM to SQL	<p>A loading knowledge module that allows client applications to load data from Oracle BAM.</p> <p>If using an Oracle BAM loading knowledge module as a source in an interface (for example LKM Oracle BAM to SQL), the user must change the default execute on button for each mapped field in the target to staging area. If left at the default source, erroneous results may occur.</p> <p>Technologies that do not allow for a staging area, such as Oracle BAM, should not have transformations performed on them.</p> <p>It has the following options:</p> <p>DELETE_TEMPORARY_OBJECTS</p> <p>DROP_PURGE</p> <p>LAST_BAM_TASK</p>
RKM Oracle BAM	<p>A customized reverse knowledge module for Oracle BAM. It has the following options:</p> <p>GET_COLUMNS</p> <p>GET_FOREIGN_KEYS</p> <p>GET_INDEXES</p> <p>GET_PRIMARY_KEYS</p> <p>LOG_FILE_NAME</p> <p>USE_LOG</p>

[Table 34–4](#) describes the parameters used in Oracle BAM knowledge modules.

Table 34–4 Oracle BAM Knowledge Module Parameters

Parameter	Description
BATCH_SIZE	<p>The maximum number of records which are sent as a batch across from the client to the server.</p> <p>The batch size that is used to send batches from the client to the server. As larger machines are used with bigger Java Virtual Machine sizes, this parameter can be increased to improve performance.</p> <p>Default value: 1024</p>
COMMIT_SIZE	<p>The maximum number of records in a single transaction. The default, 0, means commit all input records in one transaction. A positive, nonzero, value denotes that the maximum number of records to be committed at a time.</p> <p>Negative values for this option are invalid.</p> <p>Default value: 0</p>
CREATE_TARG_TABLE	<p>Select this option to create the target data object on Oracle BAM Server.</p>
DATETIME_PATTERN	<p>This option and Locale specifications (for example, LOCALE_LANGUAGE, LOCALE_COUNTRY, and LOCALE_VARIANT) are used to construct a Java SimpleDateFormat object which is used in parsing the date and time data strings.</p> <p>See Section 33.2.2, "How to Configure DateTime Specification" for information about SimpleDateFormat.</p>

Table 34–4 (Cont.) Oracle BAM Knowledge Module Parameters

Parameter	Description
DELETE_TEMPORARY_OBJECTS	Set this option to <code>NO</code> to retain temporary objects after integration. This option is useful for debugging.
DROP_PURGE	Set this option to <code>YES</code> to not only drop the work table, but purge it as well. When a table is dropped, it is recoverable in the database's recycle bin. When the table is dropped and purged, it is permanently deleted.
GET_COLUMNS	Set to <code>Yes</code> to reverse engineer the columns.
GET_FOREIGN_KEYS	Set to <code>Yes</code> to reverse engineer the foreign keys.
GET_INDEXES	Set to <code>Yes</code> to reverse engineer the indexes.
GET_PRIMARY_KEYS	Set to <code>Yes</code> to reverse engineer the primary keys.
KEY_CONDITION	<p>Set this option to match one or more corresponding rows from source to target. Use the following operators: <code>*</code>, <code>=</code>, <code>!=</code>, <code><</code>, <code><=</code>, <code>></code>, <code>>=</code>. The match value (that is, the <code>where</code> clause value) should be supplied as the mapping value for the target data store's key field in the Diagram tab for the interface in Oracle Data Integrator Designer.</p> <p>Note that when the <code>*</code> operator is chosen as the <code>KEY_CONDITION</code> option value, all rows are deleted from the target data store, regardless of its key field's mapping value.</p>
LAST_BAM_TASK	Use this option to manage the life cycle of the Oracle BAM JDBC connection. If this task is the last Oracle BAM task in the work flow, it closes the JDBC connection; otherwise, it leaves the connection open.
LOCALE_COUNTRY	<p>The country option is a valid ISO Country Code. These codes are the upper-case, two-letter codes as defined by ISO-3166.</p> <p>This option plus <code>LOCALE_LANGUAGE</code> and <code>LOCALE_VARIANT</code> are used to construct a Java Locale object.</p>
LOCALE_LANGUAGE	<p>The language option is a valid ISO Language Code. These codes are the lower-case, two-letter codes as defined by ISO-639.</p> <p>This option plus <code>LOCALE_COUNTRY</code> and <code>LOCALE_VARIANT</code> are used to construct a Java Locale object.</p>
LOCALE_VARIANT	<p>The variant option is a vendor or browser-specific code. For example, use <code>WIN</code> for Windows, <code>MAC</code> for Macintosh, and <code>POSIX</code> for POSIX. Where there are two variants, separate them with an underscore, and put the most important one first. For example, a Traditional Spanish collation might construct a locale with parameters for language, country and variant as: <code>es</code>, <code>ES</code>, <code>Traditional_WIN</code>.</p> <p>This option plus <code>LOCALE_LANGUAGE</code> and <code>LOCALE_COUNTRY</code> are used to construct a Java Locale object.</p>
LOG_FILE_NAME	Specify when <code>USE_LOG</code> is set to <code>Yes</code> . Specify the path and file name of the log. Be sure to set this property value properly (that is, choose a location where user has write permissions) before running the reverse engineering.

Table 34–4 (Cont.) Oracle BAM Knowledge Module Parameters

Parameter	Description
NON_KEY_MATCHING	<p>Determines if the incoming non-key column values are to be compared to the non-key column values in the dimension table.</p> <p>If <code>NON_KEY_MATCHING</code> is set to <code>true</code>, if the incoming non-key column values match those in the dimension table, the row is inserted into the fact table (which is the target data store). Otherwise, that row insert fails, which might even lead to the entire transaction being rolled back (in case <code>COMMIT_SIZE</code> was set to 0). A <code>COMMIT_SIZE</code> of 1 results in only this row being rolled back and ignored, and all other row inserts progress as usual.</p> <p>If <code>NON_KEY_MATCHING</code> is set to <code>false</code> and lookup succeeds, incoming non-key column values for the dimension table are ignored.</p>
USE_LOG	Set to Yes if you want the reverse-engineering process log details in a log file. Specify the log file location using the <code>LOG_FILE_NAME</code> option.

34.5 Updating the Oracle Data Integrator External Data Source Definition

When you install the Oracle BAM integration files for Oracle Data Integrator with a correctly populated properties file, you are not required to do any other configuration in Oracle BAM. Two external data source (EDS) definitions are created during the installation process, and they are populated with the correct values to connect Oracle BAM Server with the ODI_Master and ODI_Work repositories in Oracle Data Integrator. These Oracle Data Integrator-specific EDS definitions must never be deleted.

There are cases in which you must update the Oracle Data Integrator EDS definitions:

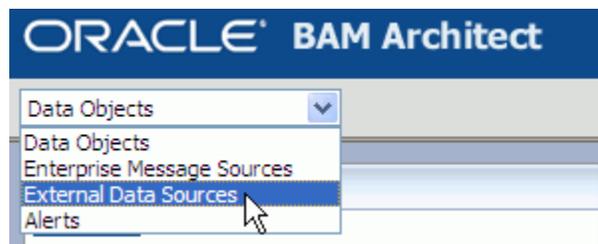
- If you change the Oracle Data Integrator login credentials, you must update the Oracle Data Integrator EDS definitions in Oracle BAM Architect.
- If the ODI_Master or ODI_Work repositories are moved to different hosts after the initial installation, you must update the corresponding EDS definitions in Oracle BAM Architect.

34.5.1 How to Update the Oracle Data Integrator External Data Source Definitions

To update the Oracle Data Integrator external data source definitions:

1. Open Oracle BAM Architect, and go to the External Data Sources page.

Figure 34–2 Opening External Data Source Page in Oracle BAM Architect



2. Select ODI_Master or ODI_Work, and click Edit.

Figure 34–3 Editing the ODI_Master External Data Source

The screenshot shows a web-based configuration interface for an External Data Source. At the top left, there are navigation links: [View](#), [Edit](#) (highlighted), [Delete](#), and [Create](#). Below these, the 'Type' is set to 'JDBC'. The 'External Data Source Name' field contains 'ODI_Master'. The 'Description' field contains the text: 'This External Data Source connects to the ODI metadata repository, and is used by the External Data Objects in /System/ODI/. Do not remove it.' The 'Driver' field contains 'oracle.jdbc.driver.OracleDriver'. The 'Login' field is empty. The 'Password' field is masked with six dots. The 'Connection String' field contains 'jdbc:oracle:thin:'. At the bottom, there are two buttons: 'Save' and 'Cancel'.

3. Update the **Login**, **Password**, or **Connection String** parameters as needed, and click **Save**.

34.6 Launching Oracle Data Integrator Scenarios From Oracle BAM Alerts

Alerts created in Oracle BAM can launch Oracle Data Integrator scenarios when specified conditions are met. See [Section F.3.9, "Run an Oracle Data Integrator Scenario"](#) for more information.

Creating External Data Sources

This chapter contains the information needed to create and manage External Data Sources (EDS).

This chapter contains the following topics:

- [Section 35.1, "Introduction to External Data Sources"](#)
- [Section 35.2, "Creating External Data Sources"](#)

35.1 Introduction to External Data Sources

An External Data Source (EDS) is a connection to an external database. An EDS usually contains data that does not change very much or data that is too large to bring into the Oracle BAM Active Data Cache (ADC).

The EDS definition in Oracle BAM acts as a pointer to the external data. For example, looking up the customer name based on a customer code in a customer management system. The customer name-code mapping is fairly static so that bringing that external data into Oracle BAM is not required.

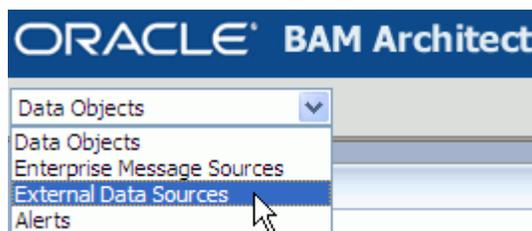
EDS definitions can be exported and imported using ICommand, but you cannot import or edit the contents using ICommand or Oracle BAM Architect.

Passwords are entered in clear text. You cannot use DSNs (data source names).

To view the existing EDS:

- Select **External Data Sources** from the Oracle BAM Architect function list.

Figure 35–1 Oracle BAM Architect Function List



35.2 Creating External Data Sources

Oracle BAM external data sources are created, edited, and deleted using Oracle BAM Architect.

35.2.1 How to Create an External Data Source

To define an EDS:

1. Select **External Data Sources** from the Oracle BAM Architect function list.
2. Click **Create**.
3. Enter a name and a description for the EDS.
4. Enter **Driver**, for example, `oracle.jdbc.driver.OracleDriver` for Oracle.
5. Enter database user credentials in the **Login** and **Password** fields.
6. Enter **Connection string/URL**, for example

```
jdbc:oracle:thin:@db_host_name:db_port:db_instance
```

35.2.2 What You May Need to Know About Oracle Data Integrator External Data Sources

If you install the integration files for Oracle BAM and Oracle Data Integrator, two EDS definitions are created in Oracle BAM Architect: ODI_Master and ODI_Work. These EDS definitions cannot be deleted from Oracle BAM Architect, and their configuration should not be changed unless you are updating your Oracle Data Integrator host.

35.2.3 How to Edit an External Data Source

To edit an EDS:

1. Select **External Data Sources** from the Oracle BAM Architect function list.
2. Select the EDS to edit.
The EDS properties display.
3. Select **Edit**.
4. Make the changes and click **Save**.

35.2.4 How to Delete an External Data Source

Note: If the EDS definitions ODI_Master and ODI_Work appear in Oracle BAM Architect, do not delete them. These EDS definitions are used by the integration between Oracle BAM and Oracle Data Integrator

To delete an EDS:

1. Select **External Data Sources** from the Oracle BAM Architect function list.
2. Select the EDS to delete.
The data source properties display.
3. Select **Delete**.
4. Click **OK** to confirm deletion of the data source.

The data source is deleted.

Using Oracle BAM Web Services

The Oracle BAM web services are part of the Oracle BAM technologies that feeds data to the Oracle BAM Server. This chapter provides information about using the Oracle BAM web services.

This chapter contains the following topics:

- [Section 36.1, "Introduction to Oracle BAM Web Services"](#)
- [Section 36.2, "Using the DataObjectOperations Web Services"](#)
- [Section 36.3, "Using the DataObjectDefinition Web Service"](#)
- [Section 36.4, "Using the ManualRuleFire Web Service"](#)
- [Section 36.5, "Using the ICommand Web Service"](#)

36.1 Introduction to Oracle BAM Web Services

The Oracle BAM web services allow users to build applications that publish data to the Oracle BAM Server for use in real-time charts and dashboards. Any client that can talk to standard web services can use these APIs to publish data to Oracle BAM. The Oracle BAM web services interfaces allow integration of Oracle BAM with other components such as Oracle BPEL Process Manager and Oracle Mediator, and they facilitate SOA composite application development.

Note: This option cannot be used for complex processing of messages, performing lookups in Oracle BAM Active Data Cache to augment the data, or initial bulk uploads to set up a star schema.

The data objects in the Oracle BAM Server are available using the Oracle BAM web services. There are several other meta objects that are available using the ICommand web service.

External web services can be called by an Oracle BAM alert rule. See [Section 37.2, "Creating Alert Rules"](#) for more information.

Oracle BAM provides the following static untyped web service APIs:

- **DataObjectOperations10131** allows clients developed for Oracle BAM 10.1.3.x servers to make web service calls to DataObjectOperations on Oracle BAM 11g servers.
- **DataObjectOperationsByID** allows developers to interact with data objects by their ID (for example, `_Call_Center`).

- **DataObjectOperationsByName** allows developers to interact with data objects by their display names (for example, Call Center).
- **DataObjectDefinition** performs operations to get, create, delete, and update definitions of Data Objects.
- **ManualRuleFire** is used by other Oracle BAM services to launch rules created in Oracle BAM Active Studio.
- **ICommand** is a DOS command-line utility that provides a set of commands that perform various operations on items in the Oracle BAM Server. The ICommand web service exposes all of the ICommand functionality through a web service.

These services can be discovered within an Oracle BAM Server using a WSIL interface.

36.2 Using the DataObjectOperations Web Services

The DataObjectOperations web service allows users to manipulate the Data Objects in the Oracle BAM Server by inserting, updating, deleting and upserting rows into the Data Objects.

The following operations are supported by the DataObjectOperations web service interfaces.

- **Batch** performs batch operations on a data object. Batch is not supported for DataObjectOperationsByName web service.
- **Delete** removes a row from the data object.
- **Get** fetches the details from a data object per the specifications in the XML payload. Get is only available in DataObjectOperationsByName web service.
- **Insert** adds a row to the data object.
- **Upsert** inserts new data into an existing row in a data object if the row exists. If the row does not exist a new row is created.
- **Update** inserts new data into an existing row in a data object.

The request and response messages vary depending on the operation used. See [Section E.1, "DataObjectOperations10131,"](#) [Section E.2, "DataObjectOperationsByName,"](#) and [Section E.3, "DataObjectOperationsByID"](#) for information about using the operations supported by each of the web services.

36.2.1 How to Use the DataObjectOperations Web Services

To use the DataObjectOperations web service, create a web service proxy in your application in Oracle JDeveloper.

The Web Services Description Language (WSDL) files for the DataObjectOperations web services are available at the following URLs on the system where Oracle BAM web services are installed.

```
http://host_name:7001/OracleBAMWS/Services/DataObject/DataObjectOperations.asmx?WSDL
```

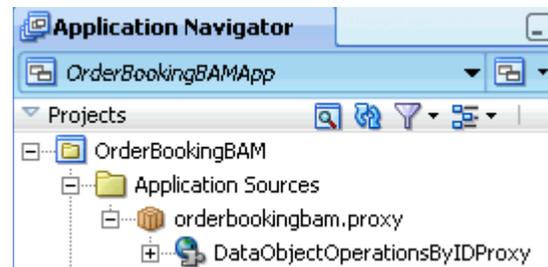
```
http://host_name:7001/OracleBAMWS/WebServices/DataObjectOperationsByID?WSDL
```

```
http://host_name:7001/OracleBAMWS/WebServices/DataObjectOperationsByName?WSDL
```

Note: The default port for Oracle BAM web services on the Administration Server is 7001. On managed servers the default port number is 9001.

When the web service proxy is created, you see it in the Application Navigator under the Application Sources folder in your project as shown in [Figure 36–1](#).

Figure 36–1 *DataObjectOperations Web service proxy in Application Sources*



36.3 Using the DataObjectDefinition Web Service

The DataObjectDefinition web service allows a web service client to create, update, delete, and get data object definitions.

The following operations are supported by DataObjectDefinition web service.

- **Create** creates a data object. For more information see [Section E.4.1, "Create."](#)
- **Delete** removes a data object from the server. For more information see [Section E.4.2, "Delete."](#)
- **Get** returns the definition of an existing data object. For more information see [Section E.4.3, "Get."](#)
- **Update** changes the definition of a data object. For more information see [Section E.4.4, "Update."](#)

The request and response messages vary depending on the operation used. See [Section E.4, "DataObjectDefinition Operations"](#) for more information.

36.3.1 How to Use the DataObjectDefinition Web Service

To use the DataObjectDefinition web service you create a web service proxy in your application in Oracle JDeveloper.

The WSDL file for the DataObjectDefinition web service is available at the following URL on the system where Oracle BAM web services are installed.

`http://host_name:7001/OracleBAMWS/WebServices/DataObjectDefinition?WSDL`

Note: The default port for Oracle BAM web services on the Administration Server is 7001. On managed servers the default port number is 9001.

When the web service proxy is created, you see it in the Application Navigator under the Application Sources folder in your project as shown in [Figure 36–2](#).

Figure 36–2 *DataObjectDefinition Web service proxy in Application Sources*



36.4 Using the ManualRuleFire Web Service

The ManualRuleFire web service allows users to launch rules in the Oracle BAM Server. FireRuleByName is the available operation. See [Section E.5, "ManualRuleFire Operations"](#) for details.

36.4.1 How to Use the ManualRuleFire Web Service

To use the ManualRuleFire web service, you create a web service proxy in your application in Oracle JDeveloper.

The WSDL file for the ManualRuleFire web service is available at the following URL on the system where Oracle BAM web services are installed.

```
http://host_name:7001/OracleBAMWS/WebServices/ManualRuleFire?WSDL
```

Note: The default port for Oracle BAM web services on the Administration Server is 7001. On managed servers the default port number is 9001.

When the web service proxy is created, you see it in the Application Navigator under the Application Sources folder in your project.

36.5 Using the ICommand Web Service

ICommand is available as a web service for application developers who want to interact with ICommand features over HTTP.

The ICommand web service includes most of the same features as the command-line utility. For example, you can use it to:

- Delete a data object
- Import rows into a data object
- Export a report

The key differences revolve around the fact that the web service cannot access files on the remote system. Therefore, you cannot pass in a file name when using the `import` command or the `export` command.

Instead, you must pass in the `import` content inline. Similarly, you receive the `export` content inline.

Commands other than `import` and `export` generally work the same as with the command-line utility.

For more information about the commands and parameters provided by ICommand, see [Appendix G, "Oracle BAM ICommand Operations and File Formats."](#)

The ICommand web service has a single method, called `Batch`. It takes a single input parameter, which is a string containing a set of commands in the syntax described in [Section G.3, "Format of Command File."](#) The return value is a string containing the results of executing each command, in the log syntax described in [Section G.4, "Format of Log File."](#)

36.5.1 How to Use the ICommand Web Service

The WSDL file for the ICommand web service is available on the system where Report Server has been installed. It is available at the following URL:

```
http://host_name:7001/OracleBAMWS/WebServices/ICommand?WSDL
```

Note: The default port for Oracle BAM web services on the Administration Server is 7001. On managed servers the default port number is 9001.

Example 36–1 *Deleting a Data Object (Input)*

```
<OracleBAMCommands>  
  <Delete type="dataobject" name="/test123"/>  
</OracleBAMCommands>
```


Creating Oracle BAM Alerts

This chapter describes how to create alerts in Oracle BAM.

This chapter contains the following topics:

- [Section 37.1, "Introduction to Creating Alerts"](#)
- [Section 37.2, "Creating Alert Rules"](#)
- [Section 37.3, "Creating Alert Rules From Templates"](#)
- [Section 37.4, "Creating Alert Rules With Messages"](#)
- [Section 37.5, "Creating Complex Alerts"](#)
- [Section 37.7, "Launching Alerts by Invoking Web Services"](#)

37.1 Introduction to Creating Alerts

Alerts are launched by a set of specified events and conditions, known as a *rule*. Alerts can be launched by data changing in a report or can be used to send a report to users daily, hourly, or at set intervals. *Events* in an alert rule can be an amount of time, a specific time, or a change in a specific report. *Conditions* restrict the alert rule to an event occurring between two specific times or dates. As a result of events and conditions, reports can be sent to users through email.

Alerts can be created in both the Oracle BAM Architect and Oracle BAM Active Studio web applications.

Alerts are shown in the Alert Rules table. In Oracle BAM Active Studio the table includes a Last Launched column that indicates the last time the alert rule was fired. Each alert name is accompanied by an icon indicating its status as described in [Table 37-1](#).

Figure 37-1 Alert Rules Table in Oracle BAM Architect

Alert Rules	
Activate	Alert Name
<input checked="" type="checkbox"/>	 Alert1
<input checked="" type="checkbox"/>	 Capacity Exceeded

Table 37–1 Alert Rule Icons

Icon	Description
	<p>Normal indicates that the alert is active and fires under the conditions specified in the rule.</p>
	<p>Invalid indicates that an alert has become orphaned or broken due to some error. This icon is displayed when an alert cannot be loaded properly into the Event Engine. The rule might require correction.</p> <p>For example, when a report is deleted and an alert based on this report still exists, that alert cannot be loaded properly.</p> <p>This icon appears only when rules are loaded into the Event Engine (on restarts). Alerts displayed with this icon do not fire again until they are edited and corrected.</p>
	<p>Expired means that the alert does not fire again. This icon is seen in time based alerts which fire only one time, after the alert has fired. However, these alerts can be edited and reused, resetting the state to Normal.</p>

Note that inactive and expired alerts behave differently. An alert can be deactivated only if it is running. This behavior is a benefit to users who do not want to receive alerts for some time interval, but want to retain the ability to activate the alert at a convenient time. Alerts that are not active, but still valid (displayed with the Normal icon) can be activated again.

Those alerts that are expired have run for the specified condition and do not run again. They cannot be activated to run again. However, if you want to reuse an expired alert, double click the alert, update the definition to make it a valid rule, and save the alert rule definition. The alert is reloaded and is ready to fire again.

Note: If any changes to the time or time zone are made on the Oracle BAM Server system, the Oracle BAM Server application must be restarted or time-based alerts misfire.

37.2 Creating Alert Rules

A rule specifies the events and conditions under which an alert fires.

Note: An alert fires only if its triggering event conditions are met from the point in time the alert is defined (or reenabled) and forward. An alert does not fire if its conditions were met before it was defined, or while it was disabled.

37.2.1 How to Create an Alert Rule

This section describes how to create Oracle BAM alert rules in Oracle BAM Architect. The procedure is the same in Oracle BAM Active Studio.

To create a rule:

1. Select **Alerts** in the Oracle BAM Architect function list.
In Oracle BAM Active Studio, select the **Alerts** tab.
2. Click **Create A New Alert**.

The Rule Creation and Edit dialog box opens.

3. Click **Create A Rule**.
4. Enter a name for the rule.
5. Select an event that launches the alert.
See [Section F.1, "Events"](#) for descriptions of each event.
6. Click **Next**.
7. Select one or more conditions, if needed.
See [Section F.2, "Conditions"](#) for descriptions of each condition.
8. Select one or more actions. See [Section F.3, "Actions"](#) for descriptions of each action.
9. In the rule expression, click each underlined item and specify a value to complete the alert rule.

For example, click **select report**, and choose a report in the dialog box that opens. Other values you define include user names receiving reports, dates and times, time intervals, and filter expressions for a specific field. To continue adding conditions or actions, click the last line in the expression and then select another condition or action.

You can click the **Back** and **Next** buttons to go between the events page and the page containing actions and conditions, and make changes to those parts of the rule expression you have constructed.

10. You can click the **Frequency Constraint** button to set a limit to how often an alert can launch.

The default frequency constraint for alerts is five seconds. Type a number and select a time measurement such as seconds, minutes, or hours, and click **OK**. To turn off the frequency constraint, uncheck the **Constraint Enabled** checkbox. For more information about frequency constraint see [Section F.4, "Frequency Constraint."](#)

11. Click **Delete this expression** to remove lines from the alert rule.
12. Click **OK**.

The alert rule is added to list and is active.

37.2.2 How to Activate Alerts

When you create an alert rule, it is automatically active. If you want an alert to be temporarily inactive but you do not want to delete it, you can turn it off by deselecting the **Activate** checkbox.

To change the activity status of an alert rule:

1. Select **Alerts** from the Oracle BAM Architect function list.
2. Select the **Activate** checkbox for the alert rule.

A checked box means the alert rule is active.

An unchecked box means the alert rule is inactive.

Selecting the **Activate** checkbox does not cause an alert to launch, it only enables the rule so that if the specified event occurs, the alert launches.

An exclamation mark on the alert icon indicates it has launched and is not valid again, or because items that it references are missing and it cannot launch.

37.2.3 How to Modify Alert Rules

When you modify alert rules created from a template, you can add new lines and select conditions and actions the same as when you build alert rules without templates.

To modify an alert rule:

1. Select the alert rule to edit.
2. Click **Edit** in the Alert Actions list.
The Rule Creation and Edit dialog box opens.
3. Make changes to the alert and click **OK**.

37.2.4 How to Delete an Alert

To delete an alert:

1. Select the alert to delete.
2. Click **Delete** in the Alert Actions list.
A dialog box opens to confirm alert deletion.
3. Click **OK**.
The alert is deleted.

37.3 Creating Alert Rules From Templates

Alert rule templates are a convenient preselected group of events and conditions based on some common use cases.

37.3.1 How to Create Alert Rules From Templates

To create an alert rule from a template:

1. Click **Create A New Alert**.
The Create Alert Rule dialog box opens.
2. Click **Create A Rule From A Template**.
3. Enter a name for the alert rule.
4. Select a template from the list.
5. In the **Rule Expression** box, click each underlined item and specify a value to complete the alert rule. For example, click **select report**, and choose a report in the dialog box that opens. Other values you define include user names receiving reports, dates and times, time intervals, and filter expressions for a specific field.
6. You can click **Frequency Constraint** to specify how often an alert can launch. The default frequency constraint for alerts is five seconds. Enter a number and select a time measurement such as seconds, minutes, or hours, and click **OK**.

7. You can click **Modify this rule** to modify the rule without using the template. This provides more options for creating rules.
8. Click **OK**.

The alert rule is added to list and is active.

37.4 Creating Alert Rules With Messages

You can create alert rules that send messages. The messages can contain information such as report names, links to reports, and user names. Messages can also include variables that are set when the alert is launched, such as the time that an event occurred and the data that launched the event. To use data variables, the event must be based on data.

37.4.1 How to Create an Alert Rule With a Message

You can create alert rules that send messages. The messages can contain information such as report names, links to reports, and user names. Messages can also include variables that are set when the alert is launched, such as the time that an event occurred and the data that launched the event. To use data variables, the event must be based on data.

To create an alert rule that includes a message:

1. Start building an alert rule.
2. Select the action **Send a message via email**.
3. Click **create message** in the rule expression.
The Alert Message dialog box opens.
4. Enter a subject in the **Subject** line.
5. Enter the message in the **Message Text** box.
6. Include special fields into the message.

Special fields are listed in the box in the lower left corner of the Alert Message dialog box. The special fields listed change when reports are selected on the right side of the dialog box.

To insert a special field into the message:

- a. Select a special field from the list.
- b. Click **Insert into subject** or **Insert into text**.

You can insert multiple values of the same type, for example, multiple links to different reports.

- **Send Report Name** inserts name of selected report.
- **Send Report Owner** inserts owner name of selected report.
- **Send Report Link** inserts link to selected report.
- **Changed Report Name** inserts name of the changed report.
- **Changed Report Owner** inserts Owner Name Of Changed Report.
- **Target User** inserts user name of message recipient.
- **Date/Time Sent** inserts date and time of message sent.

7. Click **OK**.

37.5 Creating Complex Alerts

You can create nested rules with many actions and chained rules that launch other rules.

You can chain rules by creating two types of rules:

- A dependent rule that must be launched by another rule.
- A rule with an action to launch a dependent rule.

37.5.1 How to Create a Dependent Rule

To create dependent rules:

1. Create a rule that includes the event **When this rule is launched**. No value is required for this event.
2. Create a rule that includes the action **Launch a rule** or **Launch rule if an action fails**. The **Launch rule if action fails** applies to any of the actions contained in the rule.
3. Click **select rule** in the action.
The Select Dependent Rule dialog box opens.
4. Select a dependent rule. Only rules that include the **When this rule is launched** event are displayed in the list.
5. Click **OK**.

To handle a failing action, add the action **Launch rule if action fails**. For example, if a rule is supposed to send a message, and for some reason the message does not send, you could launch another rule to notify you.

37.6 Using Alert History

This functionality is only available in Oracle BAM Active Studio.

37.6.1 How to View Alert History

You can view recent history of alert activity on the Alerts tab. The Alerts History list displays the 25 most recent alerts launched.

In the case of email alerts, the Alerts History list only displays the alert if the user logged in is an alert recipient. It is not listed in the Alerts History list—even if the user is the creator of the alert—is not a recipient of the alert.

To view the alert history:

In the Alerts History list, you can view recently launched alerts, the user who created the alerts, and the time and date that the alerts launched. Alerts that included report links in them provide links to the report from the report history list.

37.6.2 How to Clear Alert History

When many alerts are actively launching and the alert history list becomes long, you might want to clear your alert history list.

To clear the alert history:

1. On the Alerts tab, click **Clear alert history**.

A message is displayed to confirm to clear alert history.

2. Click **OK**.

The alert history list is deleted. New alerts launched after clearing appear in the alert history list.

37.7 Launching Alerts by Invoking Web Services

You can use the alerts web service to manually launch alerts. For more information, refer to:

`http://host:http_port/OracleBAMWS/WebServices/ManualRuleFire?wsdl`

You define the rule name using the format:

`username.alertname`

Note: Oracle BAM Active Studio URLs used in alerts and report links contain a virtual directory using the product build number for caching and performance purposes. This directory must be included in links, and it is not recommended to edit these links. Links created with a previous version of Oracle BAM do not work after a product upgrade. The alert requires editing or the report shortcut must be copied again.

Using ICommand

This chapter provides usage information for the ICommand command-line utility. It contains the following topics:

- [Section 38.1, "Introduction to ICommand"](#)
- [Section 38.2, "Executing ICommand"](#)
- [Section 38.3, "Specifying the Command and Option Syntax"](#)
- [Section 38.4, "Using Command-line-only Parameters"](#)
- [Section 38.5, "Running ICommand Remotely"](#)

38.1 Introduction to ICommand

ICommand is a command-line utility (and web service) that provides a set of commands that perform various operations on items in the Active Data Cache. You can use ICommand to export, import, rename, clear, and delete items from Active Data Cache. The commands can be contained in an input XML file, or a single command can be entered on the command line. Informational and error messages may be output to either the command window or to an XML file.

For more information about using the ICommand web service, see [Section 36.5, "Using the ICommand Web Service."](#)

For information about individual commands and their parameters see [Appendix G, "Oracle BAM ICommand Operations and File Formats."](#)

38.2 Executing ICommand

ICommand can be executed using the `ORACLE_HOME\bam\bin\icommand.bat` file on the Microsoft Windows platform and `ORACLE_HOME\bam\bin\icommand.sh` shell script on UNIX platforms.

Just entering `icommand` on the command line provides the user with a summary of the ICommand operations and parameters.

Before attempting to execute ICommand, the `JAVA_HOME` environment variable must be set to point to the root directory of the supported version of Java Development Kit (see the Oracle BAM support matrix on Oracle Technology Network web site for supported JDK versions).

Note: When Oracle BAM is installed, ICommand looks for the Oracle BAM Server on port 9001 by default. If the Oracle BAM Server port number is changed from the default during the setup and configuration of Oracle BAM, then the user must manually change the port number from 9001 to the new port number in the file `BAMICCommandConfig.xml`.

The property to change is

```
<ADCServerPort>9001</ADCServerPort>
```

The `BAMICCommandConfig.xml` file is located in `ORACLE_HOME\bam\config`.

38.3 Specifying the Command and Option Syntax

The basic structure of the ICommand command line entry is as follows:

```
icommand -username user_name -cmd command_name -name value -type value [-parameter value]
```

All parameters given on the command line are in the following form:

```
-parameter value
```

The `parameter` portion is not case sensitive. If the `value` portion contains spaces or other special characters, it must be enclosed in double quotation marks. For example

```
icommand -cmd export -name "/Samples/Call Center" -type dataobject
-file C:\CallCenter.xml
```

It is required to use quotation marks around report names and file names that contain spaces and other special characters.

For some parameters, the `value` may be omitted. See [Section G.2, "Detailed Operation Descriptions,"](#) for information about individual parameter values.

38.3.1 How to Specify the Security Credentials

ICommand requires users to provide security credentials when running operations. If no security credentials have been specified in the configuration file, ICommand securely prompts for a user name and password.

To use default credentials, add the `ICommand_Default_User_Name` and `ICommand_Default_Password` properties to the `ORACLE_HOME\bam\config\BAMICCommandConfig.xml` file. For example:

```
<ICommand_Default_User_Name>user_name</ICommand_Default_User_Name>
<ICommand_Default_Password>password</ICommand_Default_Password>
```

However, command line entries always override the properties specified in the configuration file.

The user name and password for running ICommand operations can come from the configuration file, command line prompts, or command line options as follows:

- If the user name and password are only specified in the configuration file (that is, `-username` parameter is not used in the command line), then the `ICommand_Default_User_Name` and `ICommand_Default_Password` values in the configuration file are used.

- If only the user name is specified in the configuration file and the password is not, then the user name value is used, and ICommand prompts the user for the password at the command line.
- If user name is specified on the command line, then that value is used, and ICommand prompts the user for a password. The password prompt occurs regardless of any properties specified in the configuration file. For example:

```
icommand -cmd export -name TestDO -file C:\TestDO.xml -username user_name
```

38.3.2 How to Specify the Command

On the command line, commands are specified by the value of the `cmd` parameter. Options for the command are specified by additional parameters. For example

```
icommand -cmd export -name TestDO
-type dataobject -file C:\TestDO.xml
```

In an XML command file, commands are specified by the XML tag. Options for the command are given as XML attribute values of the command tag, in the form `parametername=value`.

Command names and parameter values (except for Active Data Cache item names) are not case sensitive.

For information about individual commands and their parameters see [Appendix G, "Oracle BAM ICommand Operations and File Formats."](#)

38.3.3 How to Specify Object Names

Whenever an object name is specified in a command, the following rules apply.

General rules

When specified on a command line, if the name contains spaces or characters that have special meaning to DOS or UNIX, the name must be quoted according to the rules for command lines.

When specified in an XML command file, if the name contains characters that have special meaning within XML, the standard XML escaping must be used.

Data Objects

If the Data Object is not at the root, the full path name must be given, as in the following example:

```
/MyFolder/MySubfolder/MyDataObject
```

If the Data Object is at the root, the leading slash (/) is optional. The following two examples are equivalent:

```
/MyDataObject
MyDataObject
```

Data Object Folders

To specify a folder in Data Objects you must include the prefix `/public/DataObject/` at the beginning of the path to the folder.

```
/public/DataObject/MyFolder/MySubfolder
```

Reports and Report Folders

The full path name plus the appropriate prefix must be specified as in the following examples.

For shared reports the `/public/Report/` prefix must be included as shown here:

```
"/public/Report/Subfolder1/My Report"
```

For private reports the `/private:user_name/Report/` prefix must be included:

```
"/private:jsmith/Report/Subfolder1/My Report"
```

The `/private:user_name/` part of the prefix may be omitted if the user running ICommand is the user that owns the report.

```
"Report/Subfolder1/My Report"
```

The path information without the `public` or `private` prefix is saved in the export file.

Similarly, a report folder can be specified using the appropriate prefix.

```
/public/Report/Subfolder1
```

```
/private:jsmith/Report/Subfolder1
```

Alert Rules

Either the name of the Alert, or the full name of the Alert may be specified. The following two examples are equivalent for Alerts if the user running ICommand is the user that owns Alert1:

```
Alert1
```

```
/private:user_name/Rule/Alert1
```

If the user running ICommand is not the owner of Alert1, then only the second form may be used.

All other object types

Specify the full name of the object.

38.3.4 How to Specify Multiple Parameter Targets

Instead of creating a separate command line for each Active Data Cache object type, such as Dataobject, Folder, Report, and Rule, on which to execute a particular command, ICommand enables you to pass parameter values to several object types in the same command line.

For example:

```
icommand -cmd export -type all -report,rule,folder:owner 1  
-dataobject,folder:permissions 1 -systemobjects 1 -file filename.xml
```

In this example, while exporting all of the objects in the system, the command passes `owner = 1` to the report, rule, and folder Active Data Cache object types. The command also passes `permissions = 1` to the dataobject and folder object types. The comma (,) separates the object types and the parameter is listed after a colon (:).

Supplying multiple values in the example single command line gives the same results as the following three commands:

```
icommand -cmd export -type report -owner 1 ...
icommand -cmd export -type rule -owner 1 ...
icommand -cmd export -type folder -owner 1 ...
```

38.4 Using Command-line-only Parameters

The following parameters can appear only on the command line:

- **Cmd**

`-cmd commandname`

Optional parameter that specifies a single command to be executed. Any parameters needed for the command must also be on the command line.

The `Cmdfile` and `cmd` parameters are mutually exclusive. Exactly one of them must be present.

- **Cmdfile**

`-cmdfile file_name`

Optional parameter that specifies the name of the file that contains commands to be processed. Because this is an XML file, it would usually have the XML extension, although that is not required.

The `Cmdfile` and `cmd` parameters are mutually exclusive. Exactly one of them must be present.

- **Debug**

`-debug flag`

Optional parameter that indicates whether extra debugging information is to be output if there is an error. Any value other than 0 (zero), or the absence of any value, indicates that debugging information is to be output. If this parameter is not present, no debugging information is output.

- **Domain**

`-domain domain_name`

Optional parameter that specifies the domain name to use to login to the Active Data Cache (the name of the computer on which the Active Data Cache server is running).

If this parameter is omitted, `main` is used, which means the server information is obtained from the `ADCServerName` key in the `ICommand.exe.config` file.

If the reserved value `ADCInProcServer` is used, then `ICommand` directly accesses the Active Data Cache database (which must be local on the same system on which `ICommand` is running) rather than contacting the Active Data Cache server. This option is necessary **only** when the Active Data Cache server is not running; otherwise corruption of the database could occur. The information about the location and structure of the Active Data Cache database is obtained from various keys in the `ICommand.exe.config` file.

- **Logfile**

`-logfile file_name`

Optional parameter that specifies the name of the file to which results and errors are logged. If the file does not exist, it is created. If the file does exist, any contents are overwritten. Because this is an XML file, it would usually have the XML extension, although that is not required.

If this parameter is not present, results and errors are output to the console.

See [Section G.4, "Format of Log File"](#) for more information about the log file format.

- Logmode

`-logmode mode`

Optional parameter that indicates whether an existing log file is to be overwritten or appended to. The possible values for this parameter are `append` or `overwrite`. In either case, if the log file does not exist it is created.

If this parameter is not present, `overwrite` is assumed.

Note that because it is XML that is being added to the log file, if the `append` option is used the XML produced may not be strictly legal, as there is no top level root tag in the XML produced by successive appends (ICommand appends the same tag each time it is run). It is left up to the user to handle this.

- Username

`-username user_name`

Optional parameter that specifies the username that the command should run as. There is no password parameter.

ICommand requires users to specify security credentials when running commands. ICommand securely prompts for a user name and password. If the `-username` parameter is specified on the command line, ICommand prompts the user for the password only.

38.5 Running ICommand Remotely

You can run ICommand from a remote system (where Oracle BAM is installed) and execute the commands on a server located remotely. To run ICommand remotely, add the properties `ServerName` and `ServerPort` in `ORACLE_HOME\bam\config\BAMICCommandConfig.xml`, as shown below.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<BAMICCommand>
  <ADCServerName>host_name</ADCServerName>
  <ADCServerPort>7001</ADCServerPort>
  <Communication_Protocol>t3</Communication_Protocol>
  <SensorFactory>oracle.bam.common.statistics.noop.SensorFactoryImpl</SensorFactor
y>
  <GenericSatelliteChannelName>invm:topic/oracle.bam.messaging.systemobjectnotific
ation</GenericSatelliteChannelName>
</BAMICCommand>
```

The Oracle BAM version installed on the remote system should be same as the Oracle BAM Server version (that is, both servers should be from the same label).

Part VII

Using Oracle User Messaging Service

This part describes how to use Oracle User Messaging Service.

This part contains the following chapters:

- [Chapter 39, "Oracle User Messaging Service"](#)
- [Chapter 40, "Sending and Receiving Messages using the User Messaging Service Java API"](#)
- [Chapter 41, "Parlay X Web Services Multimedia Messaging API"](#)
- [Chapter 42, "User Messaging Preferences"](#)

Oracle User Messaging Service

This chapter describes Oracle User Messaging Service (UMS).

This chapter includes the following section:

- [Section 39.1, "User Messaging Service Overview"](#)

39.1 User Messaging Service Overview

Oracle User Messaging Service enables two-way communication between users and deployed applications. Key features include:

- Support for a variety of messaging channels—Messages can be sent and received through Email, IM (XMPP), SMS (SMPP), and Voice. Messages can also be delivered to a user's SOA/WebCenter Worklist.
- Two-way Messaging—In addition to sending messages from applications to users (referred to as *outbound* messaging), users can initiate messaging interactions (inbound messaging). For example, a user can send an email or text message to a specified address; the message is routed to the appropriate application which can then respond to the user or invoke another process according to its business logic.
- User Messaging Preferences—End users can use a web interface to define preferences for how and when they receive messaging notifications. Applications immediately become more flexible; rather than deciding whether to send to a user's email address or instant messaging client, the application can simply send the message to the user, and let UMS route the message according to the user's preferences.
- Robust Message Delivery—UMS keeps track of delivery status information provided by messaging gateways, and makes this information available to applications so that they can respond to a failed delivery. Or, applications can specify one or more *failover* addresses for a message in case delivery to the initial address fails. Using the failover capability of UMS frees application developers from having to implement complicated retry logic.
- Pervasive integration within Fusion Middleware: UMS is integrated with other Fusion Middleware components providing a single consolidated bi-directional user messaging service.
 - Integration with Oracle BPEL—Oracle JDeveloper includes pre-built BPEL activities that enable messaging operations. Developers can add messaging capability to a SOA composite application by dragging and dropping the necessary activity into any workflow.

- Integration with Oracle Human Workflow—UMS enables the Human Workflow engine to send actionable messages to and receive replies from users over email.
- Integration with Oracle BAM—Oracle BAM uses UMS to send email alerts in response to monitoring events.
- Integration with Oracle WebCenter—UMS APIs are available to developers building applications for Oracle WebCenter Spaces. The API is a realization of Parlay X Web Services for Multimedia Messaging, version 2.1, a standard web service interface for rich messaging.

39.1.1 Components

There are three types of components that make up Oracle User Messaging Service. These components are standard Java EE applications, making it easy to deploy and manage them using the standard tools provided with Oracle WebLogic Server.

- **UMS Server:** The UMS Server orchestrates message flows between applications and users. The server routes outbound messages from a client application to the appropriate driver, and routes inbound messages to the correct client application. The server also maintains a repository of previously sent messages in a persistent store, and correlates delivery status information with previously sent messages.
- **UMS Drivers:** UMS Drivers connect UMS to the messaging gateways, adapting content to the various protocols supported by UMS. Drivers can be deployed or undeployed independently of one another depending on what messaging channels are available in a given installation.
- **UMS Client applications:** UMS client applications implement the business logic of sending and receiving messages. A UMS client application might be a SOA application that sends messages as one step of a BPEL workflow, or a WebCenter Spaces application that can send messages from a web interface.

In addition to the components that make up UMS itself, the other key entities in a messaging environment are the external gateways required for each messaging channel. These gateways are not a part of UMS or Oracle WebLogic Server. Since UMS Drivers support widely-adopted messaging protocols, UMS can be integrated with existing infrastructures such as a corporate email servers or XMPP (Jabber) servers. Alternatively, UMS can connect to outside providers of SMS or text-to-speech services that support SMPP or VoiceXML, respectively.

39.1.2 Architecture

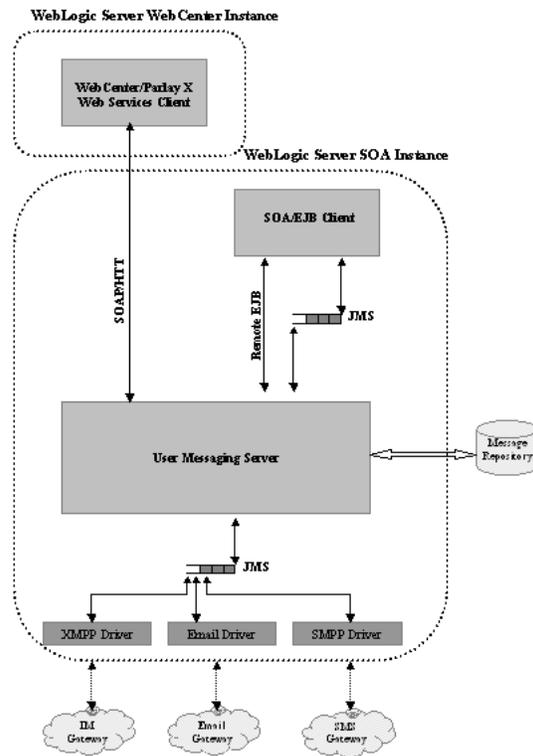
The system architecture of Oracle User Messaging Service is shown in [Figure 39–1](#).

For maximum flexibility, the components of UMS are separate Java EE applications. This allows them to be deployed and managed independently of one another. For example, a particular driver can be stopped and reconfigured without affecting message delivery on all other channels.

Exchanges between UMS client applications and the UMS Server occur as SOAP/HTTP web service requests for web service clients, or through Remote EJB and JMS calls for BPEL messaging activities. Exchanges between the UMS Server and UMS Drivers occur through JMS queues.

Oracle UMS server and drivers are installed alongside SOA or BAM in their respective WebLogic Server instances. A WebCenter installation will include the necessary libraries to act as a UMS client application, invoking a server deployed in a SOA instance.

Figure 39-1 UMS architecture



Sending and Receiving Messages using the User Messaging Service Java API

This chapter describes how to use the User Messaging Service (UMS) API to develop applications, and describes how to build two sample applications, `usermessagingsample.ear` and `usermessagingsample-echo.ear`.

This chapter includes the following sections:

- [Section 40.1, "Overview of UMS Java API"](#)
- [Section 40.2, "Creating a UMS Client Instance"](#)
- [Section 40.3, "Sending a Message"](#)
- [Section 40.4, "Receiving a Message"](#)
- [Section 40.5, "Using the UMS Enterprise JavaBeans Client API to Build a Client Application"](#)
- [Section 40.6, "Using the UMS Enterprise JavaBeans Client API to Build a Client Echo Application"](#)
- [Section 40.7, "Creating a New Application Server Connection"](#)

40.1 Overview of UMS Java API

The UMS Java API supports developing applications for Enterprise JavaBeans clients. It consists of packages grouped as follows:

- Common and Client Packages
 - `oracle.sdp.messaging`
 - `oracle.sdp.messaging.filter`: A `MessageFilter` is used by an application to exercise greater control over what messages are delivered to it.
- User Preferences Packages
 - `oracle.sdp.messaging.userprefs`
 - `oracle.sdp.messaging.userprefs.tools`

40.1.1 Creating a Java EE Application Module

There are two choices for a Java EE application module that uses the UMS Enterprise JavaBeans Client API:

- Enterprise JavaBeans Application Module - Stateless Session Bean - This is a back end, core message-receiving or message-sending application.

- Web Application Module - This is for applications that have an HTML or web front end.

Whichever application module is selected will use the UMS Client API to register the application with the UMS Server and subsequently invoke operations to send or retrieve messages, status, and register or unregister access points. For a complete list of operations refer to the UMS Javadoc.

The samples with source code are available on Oracle Technology Network (OTN).

40.2 Creating a UMS Client Instance

This section describes the requirements for creating a UMS Enterprise JavaBeans Client. You can create a `MessagingEJBClient` instance by using the code in the `MessagingClientFactory` class.

When creating an application using the UMS Enterprise JavaBeans Client, the application must be packaged as an EAR file, and the `usermessagingclient-ejb.jar` module bundled as an Enterprise JavaBeans module.

40.2.1 Creating a MessagingEJBClient Instance Using a Programmatic or Declarative Approach

[Example 40-1](#) shows code for creating a `MessagingEJBClient` instance using the programmatic approach:

Example 40-1 Programmatic Approach to Creating a MessagingEJBClient Instance

```
ApplicationInfo appInfo = new ApplicationInfo();
appInfo.setApplicationName("SampleApp");
appInfo.setApplicationInstanceName("SampleAppInstance");
MessagingClient mClient =
    MessagingClientFactory.createMessagingEJBClient(appInfo);
```

You can also create a `MessagingEJBClient` instance using a declarative approach. The declarative approach is normally the preferred approach since it allows you to make changes at deployment time.

You must specify all the required Application Info properties as environment entries in your Java EE module's descriptor (`ejb-jar.xml` or `web.xml`).

[Example 40-2](#) shows code for creating a `MessagingEJBClient` instance using the declarative approach:

Example 40-2 Declarative Approach to Creating a MessagingEJBClient Instance

```
MessagingClient mClient = MessagingClientFactory.createMessagingEJBClient();
```

40.2.2 API Reference for Class MessagingClientFactory

The API reference for class `MessagingClientFactory` can be accessed from the Javadoc.

40.3 Sending a Message

You can create a message by using the code in the `MessageFactory` class and `Message` interface of `oracle.sdp.messaging`.

The types of messages that can be created include plaintext messages, multipart messages that can consist of text/plain and text/html parts, and messages that include the creation of delivery channel (DeliveryType) specific payloads in a single message for recipients with different delivery types.

40.3.1 Creating a Message

This section describes the various types of messages that can be created.

40.3.1.1 Creating a Plaintext Message

[Example 40–3](#) shows how to create a plaintext message using the UMS Java API.

Example 40–3 Creating a Plaintext Message Using the UMS Java API

```
Message message = MessageFactory.getInstance().createTextMessage("This is a Plain
Text message.");
Message message = MessageFactory.getInstance().createMessage();
message.setContent("This is a Plain Text message.", "text/plain");
```

40.3.1.2 Creating a Multipart/Alternative Message (with Text/Plain and Text/HTML Parts)

[Example 40–4](#) shows how to create a multipart or alternative message using the UMS Java API.

Example 40–4 Creating a Multipart or Alternative Message Using the UMS Java API

```
Message message = MessageFactory.getInstance().createMessage();
MimeMultipart mp = new MimeMultipart("alternative");
MimeBodyPart mp_partPlain = new MimeBodyPart();
mp_partPlain.setContent("This is a Plain Text part.", "text/plain");
mp.addBodyPart(mp_partPlain);
MimeBodyPart mp_partRich = new MimeBodyPart();
mp_partRich
    .setContent(
        "<html><head></head><body><b><i>This is an HTML
part.</i></b></body></html>",
        "text/html");
mp.addBodyPart(mp_partRich);
message.setContent(mp, "multipart/alternative");
```

40.3.1.3 Creating Delivery Channel-Specific Payloads in a Single Message for Recipients with Different Delivery Types

When sending a message to a destination address, there could be multiple channels involved. Oracle UMS application developers are required to specify the correct multipart format for each channel.

[Example 40–5](#) shows how to create delivery channel (DeliveryType) specific payloads in a single message for recipients with different delivery types.

Each top-level part of a multiple payload multipart/alternative message should contain one or more values of this header. The value of this header should be the name of a valid delivery type. Refer to the available values for *DeliveryType* in the enum DeliveryType.

Example 40–5 Creating Delivery Channel-specific Payloads in a Single Message for Recipients with Different Delivery Types

```
Message message = MessageFactory.getInstance().createMessage();

// create a top-level multipart/alternative MimeMultipart object.
MimeMultipart mp = new MimeMultipart("alternative");

// create first part for SMS payload content.
MimeBodyPart part1 = new MimeBodyPart();
part1.setContent("Text content for SMS.", "text/plain");

part1.setHeader(Message.HEADER_NS_PAYLOAD_PART_DELIVERY_TYPE, "SMS");

// add first part
mp.addBodyPart(part1);

// create second part for EMAIL and IM payload content.
MimeBodyPart part2 = new MimeBodyPart();
MimeMultipart part2_mp = new MimeMultipart("alternative");
MimeBodyPart part2_mp_partPlain = new MimeBodyPart();
part2_mp_partPlain.setContent("Text content for EMAIL/IM.", "text/plain");
part2_mp.addBodyPart(part2_mp_partPlain);
MimeBodyPart part2_mp_partRich = new MimeBodyPart();
part2_mp_partRich.setContent("<html><head></head><body><b><i>" + "HTML content for
EMAIL/IM." +
"</i></b></body></html>", "text/html");
part2_mp.addBodyPart(part2_mp_partRich);
part2.setContent(part2_mp, "multipart/alternative");

part2.addHeader(Message.HEADER_NS_PAYLOAD_PART_DELIVERY_TYPE, "EMAIL");
part2.addHeader(Message.HEADER_NS_PAYLOAD_PART_DELIVERY_TYPE, "IM");

// add second part
mp.addBodyPart(part2);

// set the content of the message
message.setContent(mp, "multipart/alternative");

// set the MultiplePayload flag to true
message.setMultiplePayload(true);
```

40.3.2 API Reference for Class MessageFactory

The API reference for class MessageFactory can be accessed from the Javadoc.

40.3.3 API Reference for Interface Message

The API reference for interface Message can be accessed from the Javadoc.

40.3.4 API Reference for Enum DeliveryType

The API reference for enum DeliveryType can be accessed from the Javadoc.

40.3.5 Addressing a Message

This section describes type of addresses and how to create address objects.

40.3.5.1 Types of Addresses

There are two types of addresses, *device addresses* and *user addresses*. A device address can be of various types, such as email addresses, instant messaging addresses, and telephone numbers. User addresses are user IDs in a user repository.

40.3.5.2 Creating Address Objects

You can address senders and recipients of messages by using the class `AddressFactory` to create `Address` objects defined by the `Address` interface.

40.3.5.2.1 Creating a Single Address Object [Example 40–6](#) shows code for creating a single `Address` object:

Example 40–6 Creating a Single Address Object

```
Address recipient =
AddressFactory.getInstance().createAddress("Email:john.doe@oracle.com");
```

40.3.5.2.2 Creating Multiple Address Objects in a Batch [Example 40–7](#) shows code for creating multiple `Address` objects in a batch:

Example 40–7 Creating Multiple Address Objects in a Batch

```
String[] recipientsStr = {"Email:john.doe@oracle.com",
"IM:jabber|john.doe@oracle.com"};
Address[] recipients = AddressFactory.getInstance().createAddress(recipientsStr);
```

40.3.5.2.3 Adding Sender or Recipient Addresses to a Message [Example 40–8](#) shows code for adding sender or recipient addresses to a message:

Example 40–8 Adding Sender or Recipient Addresses to a Message

```
Address sender =
AddressFactory.getInstance().createAddress("Email:john.doe@oracle.com");
Address recipient =
AddressFactory.getInstance().createAddress("Email:jane.doe@oracle.com");
message.addSender(sender);
message.addRecipient(recipient);
```

40.3.5.3 Creating a Recipient with a Failover Address

[Example 40–9](#) shows code for creating a recipient with a failover address:

Example 40–9 Creating a Single Address Object with Failover

```
String recipientWithFailoverStr = "Email:john.doe@oracle.com,
IM:jabber|john.doe@oracle.com";
Address recipient =
AddressFactory.getInstance().createAddress(recipientWithFailoverStr);
```

40.3.5.4 API Reference for Class `AddressFactory`

The API reference for class `AddressFactory` can be accessed from the Javadoc.

40.3.5.5 API Reference for Interface `Address`

The API reference for interface `Address` can be accessed from the Javadoc.

40.3.6 Retrieving Message Status

You can use Oracle UMS to retrieve message status either synchronously or asynchronously.

40.3.6.1 Synchronous Retrieval of Message Status

To perform a synchronous retrieval of current status, use the following flow from the `MessagingClient` API:

```
String messageId = messagingClient.send(message);
Status[] statuses = messagingClient.getStatus(messageId);
```

or,

```
Status[] statuses = messagingClient.getStatus(messageId, address[]) --- where
address[] is an array of one or more of the recipients set in the message.
```

40.3.6.2 Asynchronous Notification of Message Status

To retrieve an asynchronous notification of message status, perform the following:

1. Implement a status listener.
2. Register a status listener (declarative way)
3. Send a message (`messagingClient.send(message);`)
4. The application automatically gets the status through an `onStatus(status)` callback of the status listener.

40.4 Receiving a Message

This section describes how an application receives messages. To receive a message you must first register an access point. From the application perspective there are two modes for receiving a message, synchronous and asynchronous.

40.4.1 Registering an Access Point

`AccessPoint` represents one or more device addresses to receive incoming messages. An application that wants to receive incoming messages must register one or more access points that represent the recipient addresses of the messages. The server matches the recipient address of an incoming message against the set of registered access points, and routes the incoming message to the application that registered the matching access point.

You can use `AccessPointFactory.createAccessPoint` to create an access point and `MessagingClient.registerAccessPoint` to register it for receiving messages.

To register an SMS access point for the number 9000:

```
AccessPoint accessPointSingleAddress =
    AccessPointFactory.createAccessPoint(AccessPoint.AccessPointType.SINGLE_ADDRESS,
        DeliveryType.SMS, "9000");
messagingClient.registerAccessPoint(accessPointSingleAddress);
```

To register SMS access points in the number range 9000 to 9999:

```
AccessPoint accessPointRangeAddress =
    AccessPointFactory.createAccessPoint(AccessPoint.AccessPointType.NUMBER_RANGE,
        DeliveryType.SMS, "9000,9999");
```

```
messagingClient.registerAccessPoint(accessPointRangeAddress);
```

40.4.2 Synchronous Receiving

You can use the method `MessagingClient.receive` to synchronously receive messages. This is a convenient polling method for light-weight clients that do not want the configuration overhead associated with receiving messages asynchronously. This method returns a list of messages that are immediately available in the application inbound queue.

It performs a nonblocking call, so if no message is currently available, the method returns null.

Note: A single invocation does not guarantee retrieval of all available messages. You must poll to ensure receiving all available messages.

40.4.3 Asynchronous Receiving

Asynchronous receiving involves a number of tasks, including configuring MDBs and writing a Stateless Session Bean message listener. See the sample application `usermessaging-sample-echo` for detailed instructions.

40.4.4 Message Filtering

A `MessageFilter` is used by an application to exercise greater control over what messages are delivered to it. A `MessageFilter` contains a matching criterion and an action. An application can register a series of message filters; they will be applied in order against an incoming (received) message; if the criterion matches the message, the action is taken. For example, an application can use `MessageFilters` to implement necessary blacklists, by rejecting all messages from a given sender address.

You can use `MessageFilterFactory.createMessageFilter` to create a message filter, and `MessagingClient.registerMessageFilter` to register it. The filter is added to the end of the current filter chain for the application. When a message is received, it is passed through the filter chain in order; if the message matches a filter's criterion, the filter's action is taken immediately. If no filters match the message, the default action is to accept the message and deliver it to the application.

For example, to reject a message with the subject "spam":

```
MessageFilter subjectFilter = MessageFilterFactory.createMessageFilter("spam",
    MessageFilter.FieldType.SUBJECT, null, MessageFilter.Action.REJECT);
messagingClient.registerMessageFilter(subjectFilter);
```

To reject messages from email address `spammer@foo.com`:

```
MessageFilter senderFilter =
    MessageFilterFactory.createBlacklistFilter("spammer@foo.com");
messagingClient.registerMessageFilter(senderFilter);
```

40.5 Using the UMS Enterprise JavaBeans Client API to Build a Client Application

This section describes how to create an application called `usermessaging-sample`, a web client application that uses the UMS Enterprise JavaBeans Client API for both

outbound messaging and the synchronous retrieval of message status. *usermessagingsample* also supports inbound messaging. Once you have deployed and configured *usermessagingsample*, you can use it to send a message to an email client.

Of the two application modules choices described in [Section 40.1.1, "Creating a Java EE Application Module,"](#) this sample focuses on the Web Application Module (WAR), which defines some HTML forms and servlets. You can examine the code and corresponding XML files for the web application module from the provided *usermessagingsample-src.zip* source. The servlets uses the UMS Enterprise JavaBeans Client API to create an UMS Enterprise JavaBeans Client instance (which in turn registers the application's info) and sends messages.

This application, which is packaged as an Enterprise ARchive file (EAR) called *usermessagingsample.ear*, has the following structure:

- `usermessagingsample.ear`
 - `META-INF`
 - `application.xml` -- Descriptor file for all of the application modules.
 - `weblogic-application.xml` -- Descriptor file that contains the import of the `oracle.sdp.messaging` shared library.
 - `usermessagingclient-ejb.jar` -- Contains the Message Enterprise JavaBeans Client deployment descriptors.
 - * `META-INF`
 - `ejb-jar.xml`
 - `weblogic-ejb.jar.xml`
 - `usermessagingsample-web.ear` -- Contains the web-based front-end and servlets.
 - * `WEB-INF`
 - `web.xml`
 - `weblogic.xml`

The prebuilt sample application, and the source code (*usermessagingsample-src.zip*) are available on OTN.

40.5.1 Overview of Development

The following steps describe the process of building an application capable of outbound messaging using *usermessagingsample.ear* as an example:

1. [Section 40.5.2, "Configuring the Email Driver"](#)
2. [Section 40.5.3, "Using JDeveloper 11g to Build the Application"](#)
3. [Section 40.5.4, "Deploying the Application"](#)
4. [Section 40.5.5, "Testing the Application"](#)

40.5.2 Configuring the Email Driver

To enable the Oracle User Messaging Service's email driver to perform outbound messaging and status retrieval, configure the email driver as follows:

- Enter the name of the SMTP mail server as the value for the `OutgoingMailServer` property.

Note: This sample application is generic and can support outbound messaging through other channels when the appropriate messaging drivers are deployed and configured.

40.5.3 Using JDeveloper 11g to Build the Application

This section describes using a Windows-based build of JDeveloper to build, compile, and deploy *usermessagingsample* through the following steps:

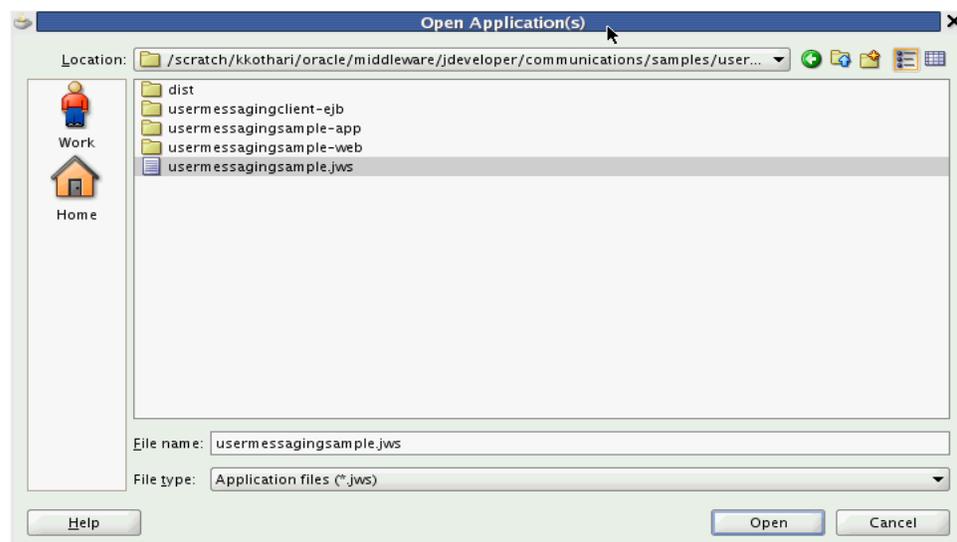
40.5.3.1 Opening the Project

1. Unzip *usermessagingsample-src.zip*, to the *JDEV_HOME/communications/samples/* directory. This directory must be used for the shared library references to be valid in the project.

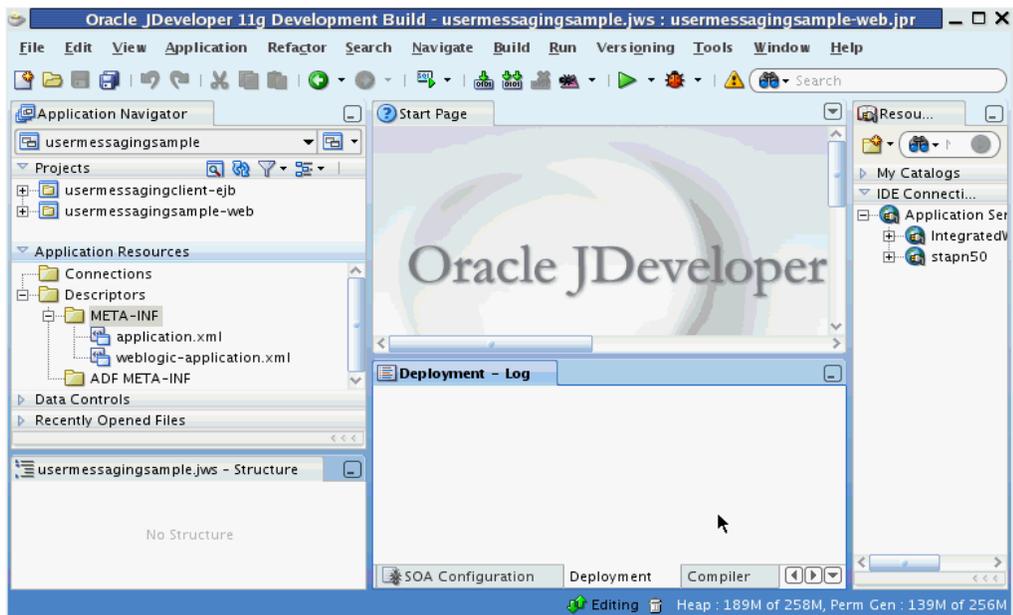
Note: If you choose to use a different directory, you must update the *oracle.sdp.messaging* library source path to *JDEV_HOME/communications/modules/oracle.sdp.messaging_11.1.1/sdpmessaging.jar*.

2. Open *usermessagingsample.jws* (contained in the .zip file) in Oracle JDeveloper.

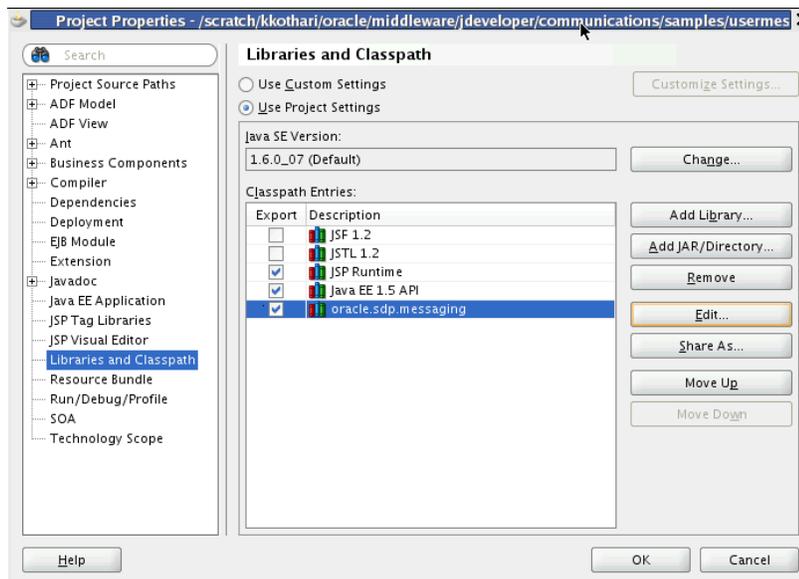
Figure 40–1 Oracle JDeveloper Main Window



In the Oracle JDeveloper main window, the project appears.

Figure 40–2 Oracle JDeveloper Main Window

3. Verify that the build dependencies for the sample application have been satisfied by checking that the following library has been added to the web module.
 - Library: `oracle.sdp.messaging`, Classpath: `JDEV_HOME/communications/modules/oracle.sdp.messaging_11.1.1/sdpMessaging.jar`. This is the Java library used by UMS and applications that use UMS to send and receive messages.
 1. In the Application Navigator, right-click web module **usermessagingsample-web**, and select **Project Properties**.
 2. In the left pane, select **Libraries and Classpath**.

Figure 40–3 Verifying Libraries

3. Click **OK**.
4. Verify that the **usermessagingclient-ejb** project exists in the application. This is an Enterprise JavaBeans module that packages the messaging client beans used by UMS applications. The module allows the application to connect with the UMS server.
5. Explore the Java files under the **usermessagingssample-web** project to see how the messaging client APIs are used to send messages, get statuses, and synchronously receive messages. The application info that is registered with the UMS Server is specified programmatically in `SampleUtils.java` in the project (Example 40–10).

Example 40–10 Application Information

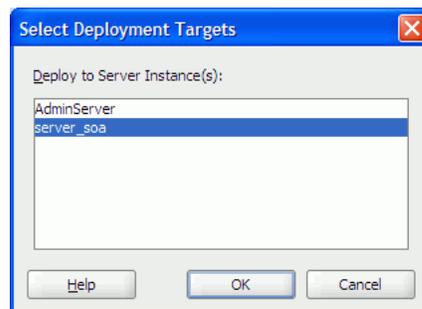
```
ApplicationInfo appInfo = new ApplicationInfo();
appInfo.setApplicationName(SampleConstants.APP_NAME);
appInfo.setApplicationInstanceName(SampleConstants.APP_INSTANCE_NAME);
appInfo.setSecurityPrincipal(request.getUserPrincipal().getName());
```

40.5.4 Deploying the Application

Perform the following steps to deploy the application:

1. Create an Application Server Connection by right-clicking the application in the navigation pane and selecting **New**. Follow the instructions in Section 40.7, "Creating a New Application Server Connection."
2. Deploy the application by selecting the **usermessagingssample** application, **Deploy**, **usermessagingssample**, **to**, and **SOA_server** (Figure 40–4).

Figure 40–4 Deploying the Project



3. Verify that the message `Build Successful` appears in the log.
 4. Verify that the message `Deployment Finished` appears in the deployment log.
- You have successfully deployed the application.

Before you can run the sample, you must configure any additional drivers in Oracle User Messaging Service and optionally configure a default device for the user receiving the message in User Messaging Preferences.

Note: Refer to *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite* for more information.

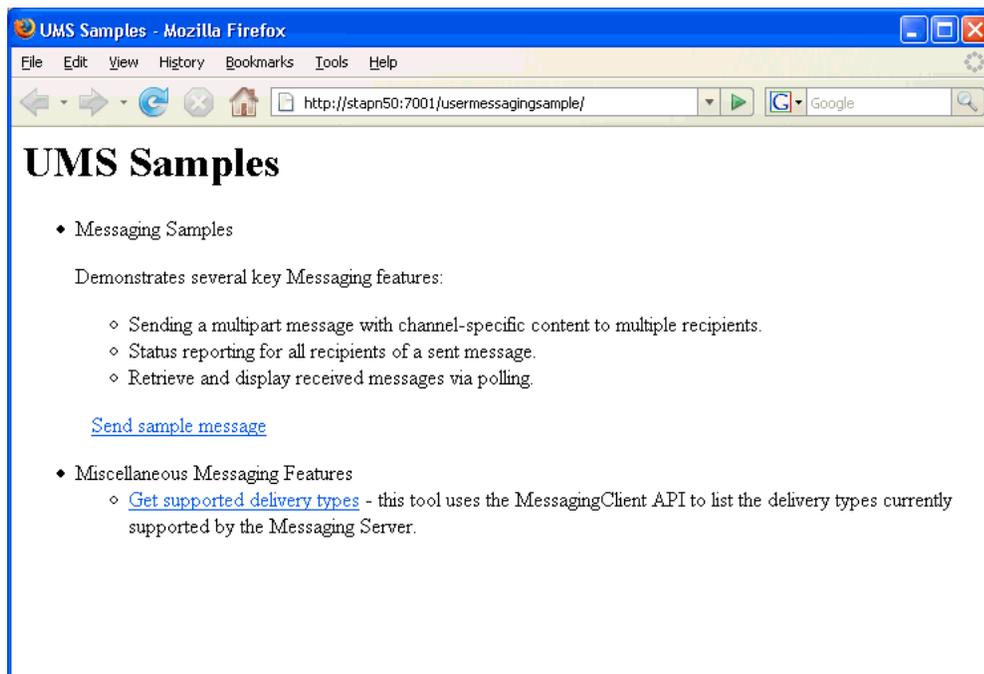
40.5.5 Testing the Application

Once **usermessagingsample** has been deployed to a running instance of Oracle WebLogic Server, perform the following:

1. Launch a web browser and enter the address of the sample application as follows: `http://host:http-port/usermessagingsample/`. For example, enter `http://localhost:7001/usermessagingsample/` into the browser's navigation bar.

When prompted, enter login credentials. For example, username `weblogic`. The browser page for testing messaging samples appears (Figure 40-5).

Figure 40-5 Testing the Sample Application



2. Click **Send sample message**. The Send Message page appears (Figure 40-6).

Figure 40–6 Addressing the Test Message

UMS Sample: Send Message - Mozilla Firefox

File Edit View History Bookmarks Tools Help

http://stapn50:7001/usermessagingsample/sample.html

UMS Sample: Send Message

Enter Sender Addresses (optional):
[e.g. "IM:sender@example.com"]
Separate multiple addresses using comma.
Note: If you enter sender addresses they will also be registered as access points with the UMS. Replies sent by the recipient to one of these addresses will be routed to this application. In this sample, there is an option to poll the receiving queue to retrieve such received messages. (This assumes that the underlying Messaging Driver used is capable of and configured for two-way messaging.)

EMAIL: test@oracle.com

Enter Recipient Addresses:
[e.g. "IM:recipient@example.com"]
Separate multiple addresses using comma.

EMAIL: john.doe@oracle.com

Enter a Subject (optional):

hello

Channel specific payload 1
Select Delivery Types:

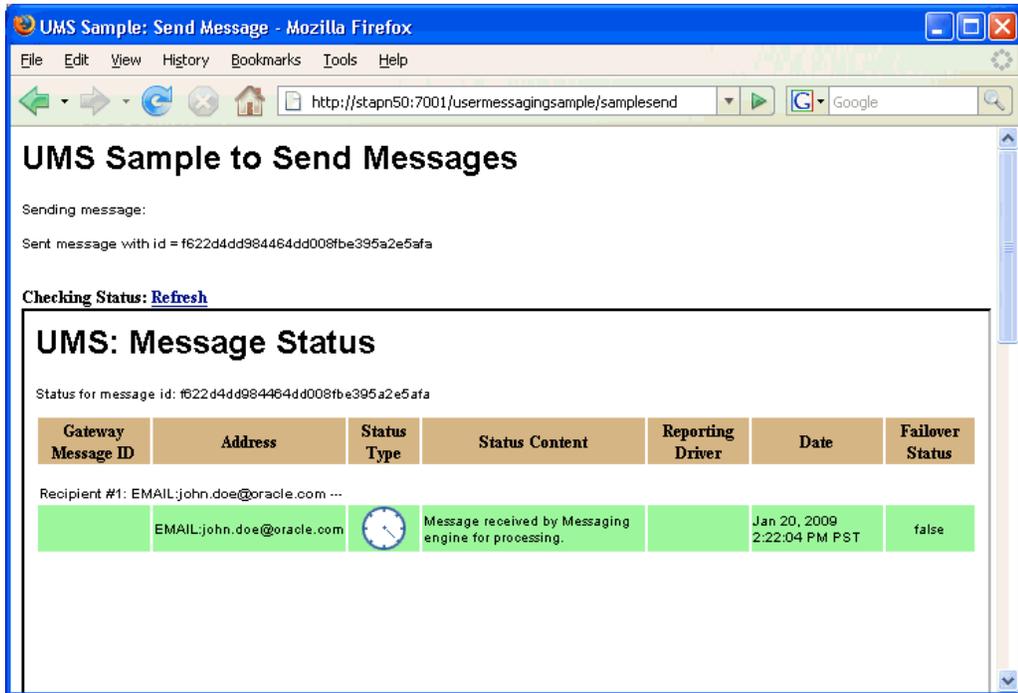
SMS
EMAIL
IM
VOICE
TWO_WAY_PAGER

Content Type:
text/plain; charset=UTF-8

Message Content:
This is a sample message.

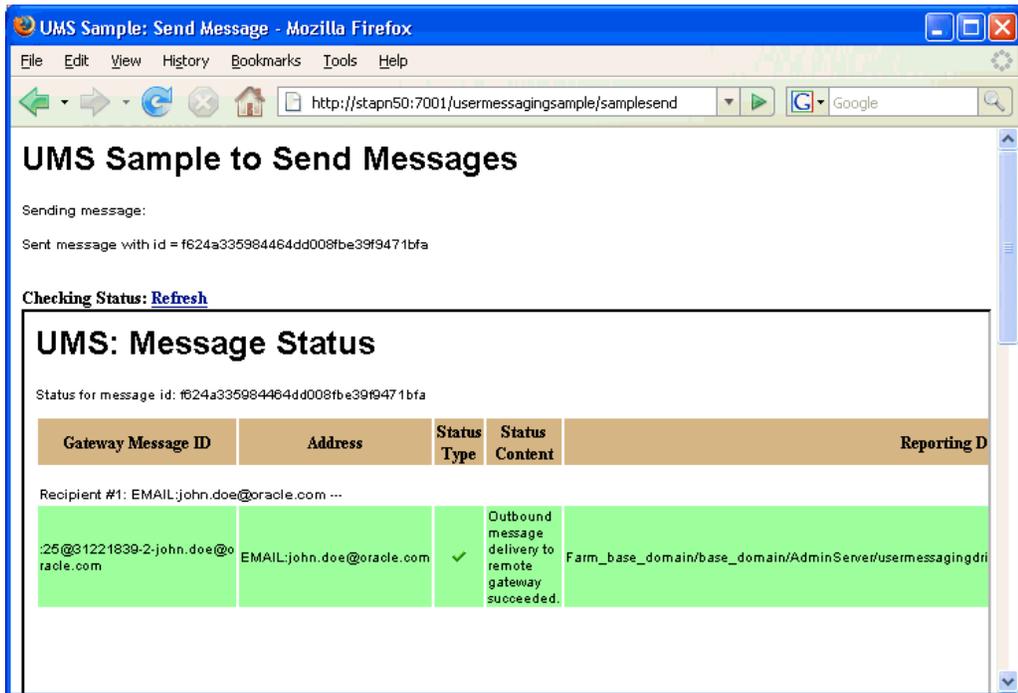
3. As an optional step, enter the sender address in the following format:
Email: sender_address.
For example, enter *Email: sender@oracle.com.*
4. Enter one or more recipient addresses. For example, enter *Email: recipient@oracle.com.* Enter multiple addresses as a comma-separated list as follows:
Email: recipient_address1, Email: recipient_address2.
If you have configured user messaging preferences, you can address the message simply to *User: username.* For example, *User: weblogic.*
5. As an optional step, enter a subject line or content for the email.
6. Click **Send**. The Message Status page appears, showing the progress of transaction (Message received by Messaging engine for processing in [Figure 40–7](#)).

Figure 40–7 Message Status



7. Click **Refresh** to update the status. When the email message has been delivered to the email server, the *Status Content* field displays *Outbound message delivery to remote gateway succeeded.*, as illustrated in Figure 40–8.

Figure 40–8 Checking the Message Status



40.6 Using the UMS Enterprise JavaBeans Client API to Build a Client Echo Application

This section describes how to create an application called `usermessagingsample-echo`, a demo client application that uses the UMS Enterprise JavaBeans Client API to asynchronously receive messages from an email address and echo a reply back to the sender.

This application, which is packaged as a Enterprise Archive file (EAR) called `usermessagingsample-echo.ear`, has the following structure:

- `usermessagingsample-echo.ear`
 - META-INF
 - `application.xml` -- Descriptor file for all of the application modules.
 - `weblogic-application.xml` -- Descriptor file that contains the import of the `oracle.sdp.messaging` shared library.
 - `usermessagingclient-ejb.jar` -- Contains the Message Enterprise JavaBeans Client deployment descriptors.
 - * META-INF
 - `ejb-jar.xml`
 - `weblogic-ejb-jar.xml`
 - `usermessagingsample-echo-ejb.jar` -- Contains the application session beans (`ClientSenderBean`, `ClientReceiverBean`) that process a received message and return an echo response.
 - * META-INF
 - `ejb-jar.xml`
 - `weblogic-ejb-jar.xml`
 - `usermessagingsample-echo-web.war` -- Contains the web-based front-end and servlets.
 - * WEB-INF
 - `web.xml`
 - `weblogic.xml`

The prebuilt sample application, and the source code (`usermessagingsample-echo-src.zip`) are available on OTN.

40.6.1 Overview of Development

The following steps describe the process of building an application capable of asynchronous inbound and outbound messaging using `usermessagingsample-echo.ear` as an example:

1. [Section 40.6.2, "Configuring the Email Driver"](#)
2. [Section 40.6.3, "Using JDeveloper 11g to Build the Application"](#)
3. [Section 40.6.4, "Deploying the Application"](#)
4. [Section 40.6.5, "Testing the Application"](#)

40.6.2 Configuring the Email Driver

To enable the Oracle User Messaging Service's email driver to perform inbound and outbound messaging and status retrieval, configure the email driver as follows:

- Enter the name of the SMTP mail server as the value for the **OutgoingMailServer** property.
- Enter the name of the IMAP4/POP3 mail server as the value for the **IncomingMailServer** property. Also, configure the incoming user name, and password.

Note: This sample application is generic and can support inbound and outbound messaging through other channels when the appropriate messaging drivers are deployed and configured.

40.6.3 Using JDeveloper 11g to Build the Application

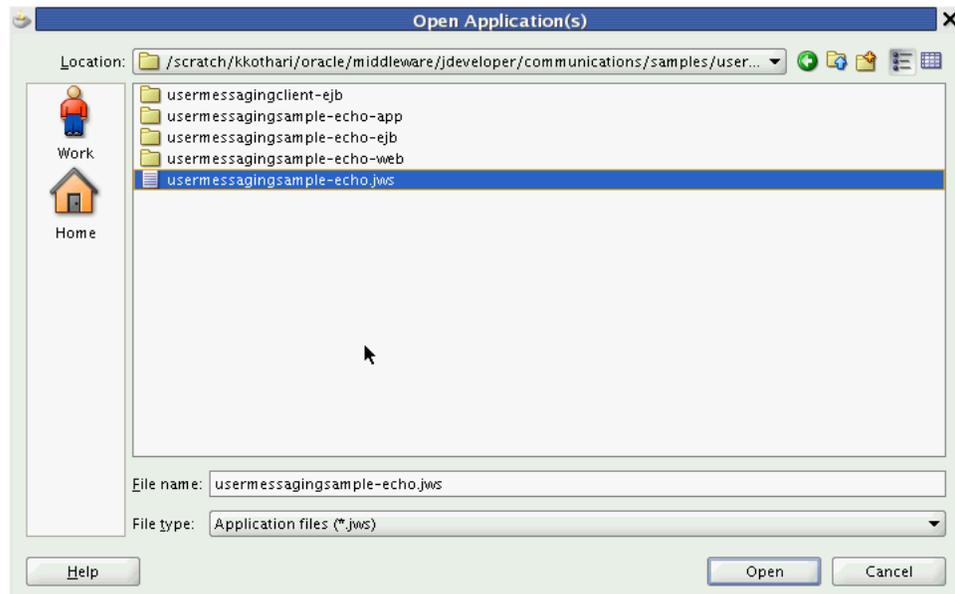
This section describes using a Windows-based build of JDeveloper to build, compile, and deploy **usermessaging-sample-echo** through the following steps:

40.6.3.1 Opening the Project

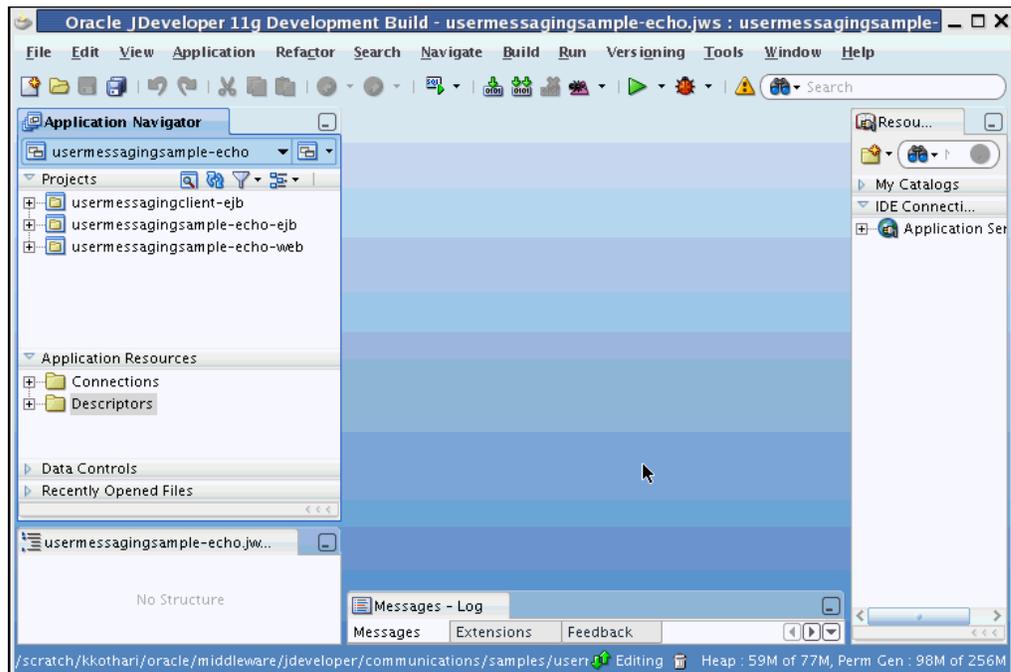
1. Unzip **usermessaging-sample-echo-src.zip**, to the *JDEV_HOME/communications/samples/* directory. This directory must be used for the shared library references to be valid in the project.

Note: If you choose to use a different directory, you must update the **oracle.sdp.messaging** library source path to *JDEV_HOME/communications/modules/oracle.sdp.messaging_11.1.1/sdpMessaging.jar*.

2. Open **usermessaging-sample-echo.jws** (contained in the .zip file) in Oracle JDeveloper ([Figure 40-9](#)).

Figure 40–9 Opening the Project

In the Oracle JDeveloper main window the project appears (Figure 40–10).

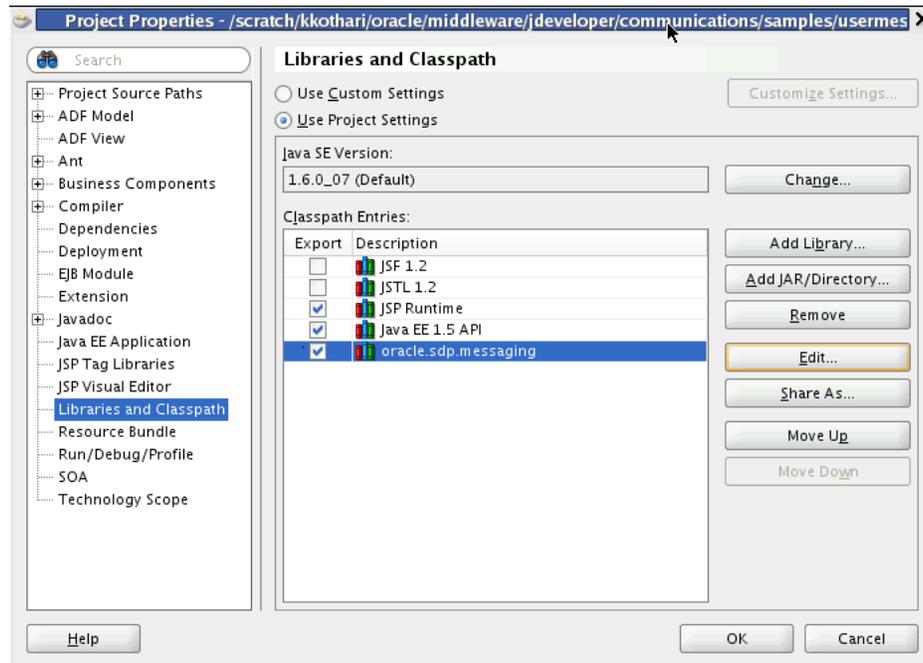
Figure 40–10 Oracle JDeveloper Main Window

3. Verify that the build dependencies for the sample application have been satisfied by checking that the following library has been added to the `usermessagingsample-echo-web` and `usermessagingsample-echo-ejb` modules.
 - Library: `oracle.sdp.messaging`, Classpath: `JDEV_HOME/communications/modules/oracle.sdp.messaging_11.1.1/sdpmessaging.jar`. This is the Java library used by UMS and applications that use UMS to send and receive messages.

Perform the following steps for each module:

1. In the Application Navigator, right-click the module and select **Project Properties**.
2. In the left pane, select **Libraries and Classpath** (Figure 40–11).

Figure 40–11 Verifying Libraries



3. Click **OK**.
4. Verify that the **usermessagingclient-ejb** project exists in the application. This is an Enterprise JavaBeans module that packages the messaging client beans used by UMS applications. The module allows the application to connect with the UMS server.
5. Explore the Java files under the **usermessagingsample-echo-ejb** project to see how the messaging client APIs are used to asynchronously receive messages (`ClientReceiverBean`), and send messages (`ClientSenderBean`).
6. Explore the Java files under the **usermessagingsample-echo-web** project to see how the messaging client APIs are used to register and unregister access points.
7. Note that the application info that is registered with the UMS Server is specified declaratively in the **usermessagingclient-ejb** project's `ejb-jar.xml` file. (Example 40–11).

Example 40–11 Application Information

```

<env-entry>
  <env-entry-name>sdpm/ApplicationName</env-entry-name>
  <env-entry-type>java.lang.String</env-entry-type>
  <env-entry-value>UMSEchoApp</env-entry-value>
</env-entry>
<env-entry>
  <env-entry-name>sdpm/ApplicationInstanceName</env-entry-name>
  <env-entry-type>java.lang.String</env-entry-type>
  <env-entry-value>UMSEchoAppInstance</env-entry-value>

```

```

</env-entry>

<env-entry>
  <env-entry-name>sdpm/ReceivingQueuesInfo</env-entry-name>
  <env-entry-type>java.lang.String</env-entry-type>

<env-entry-value>OraSDPM/QueueConnectionFactory:OraSDPM/Queues/OraSDPMAppDefRcvQ1<
/entry-value>
</env-entry>

<env-entry>
  <env-entry-name>
    sdpm/MessageListenerSessionBeanJNDIName
  </env-entry-name>
  <env-entry-type>java.lang.String</env-entry-type>
  <env-entry-value>
    ejb/umsEchoApp/ClientReceiverLocal</env-entry-value>
</env-entry>
<env-entry>
  <env-entry-name>
    sdpm/MessageListenerSessionBeanHomeClassName</env-entry-name>
  <env-entry-type>java.lang.String</env-entry-type>
  <env-entry-value>
    oracle.sdp.messaging.sample.ejbApp.ClientReceiverHomeLocal
  </env-entry-value>
</env-entry>
<env-entry>
  <env-entry-name>
    sdpm/StatusListenerSessionBeanJNDIName
  </env-entry-name>
  <env-entry-type>java.lang.String</env-entry-type>

<env-entry-value>ejb/umsEchoApp/ClientReceiverLocal</env-entry-value>
</env-entry>
<env-entry>

<env-entry-name>sdpm/StatusListenerSessionBeanHomeClassName</env-entry-name>
  <env-entry-type>java.lang.String</env-entry-type>

<env-entry-value>oracle.sdp.messaging.sample.ejbApp.ClientReceiverHomeLocal</env-
entry-value>
</env-entry>

```

8. Note that the Application Name (UMSEchoApp) and Application Instance Name (UMSEchoAppInstance) are also used in the Message Selector for the MessageDispatcherBean MDB, which is used for asynchronous receiving of messages and statuses placed in the application receiving queue ([Example 40-12](#)).

Example 40-12 Application Information

```

<activation-config-property>
  <activation-config-property-name>
    messageSelector
  </activation-config-property-name>
  <activation-config-property-value>
    appName='UMSEchoApp' or sessionName='UMSEchoApp-UMSEchoAppInstance'
  </activation-config-property-value>
</activation-config-property>

```

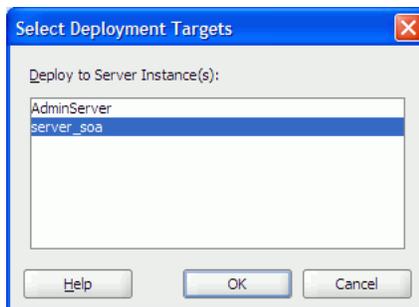
Note: If you chose a different Application Name and Application Instance Name for your own application, remember to update this message selector. Asynchronous receiving does not work otherwise.

40.6.4 Deploying the Application

Perform the following steps to deploy the application:

1. Create an Application Server Connection by right-clicking the application in the navigation pane and selecting New. Follow the instructions in [Section 40.7, "Creating a New Application Server Connection."](#)
2. Deploy the application by selecting the **usermessagingsample-echo** application, **Deploy**, **usermessagingsample-echo**, **to**, and **SOA_server** ([Figure 40–12](#)).

Figure 40–12 Deploying the Project



3. Verify that the message `Build Successful` appears in the log.
4. Verify that the message `Deployment Finished` appears in the deployment log.

You have successfully deployed the application.

Before you can run the sample you must configure any additional drivers in Oracle User Messaging Service and optionally configure a default device for the user receiving the message in User Messaging Preferences.

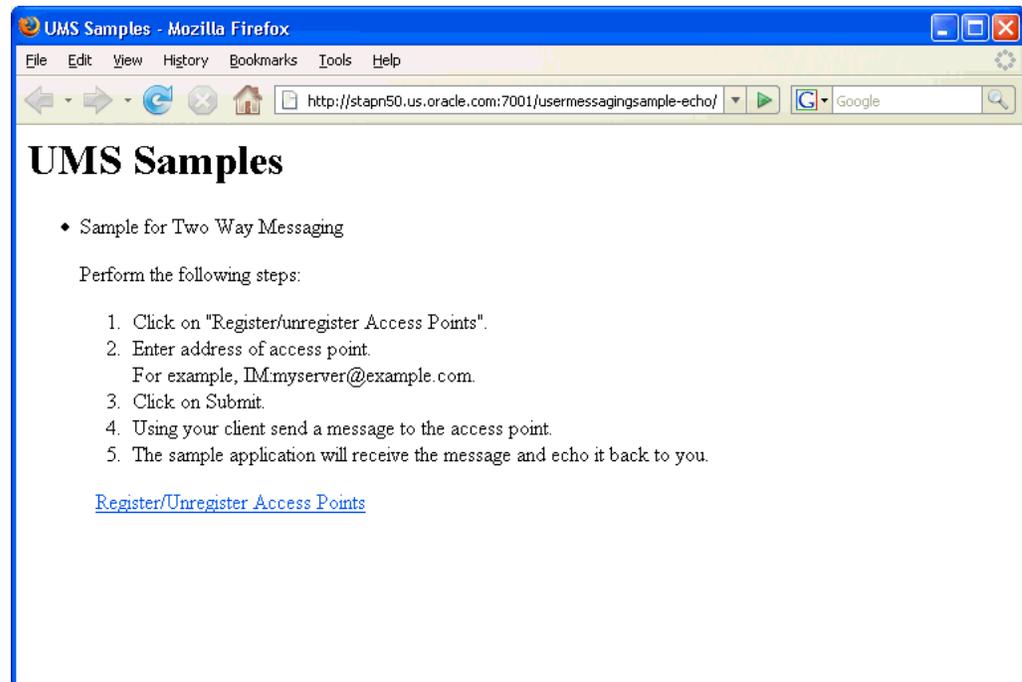
Note: Refer to *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite* for more information.

40.6.5 Testing the Application

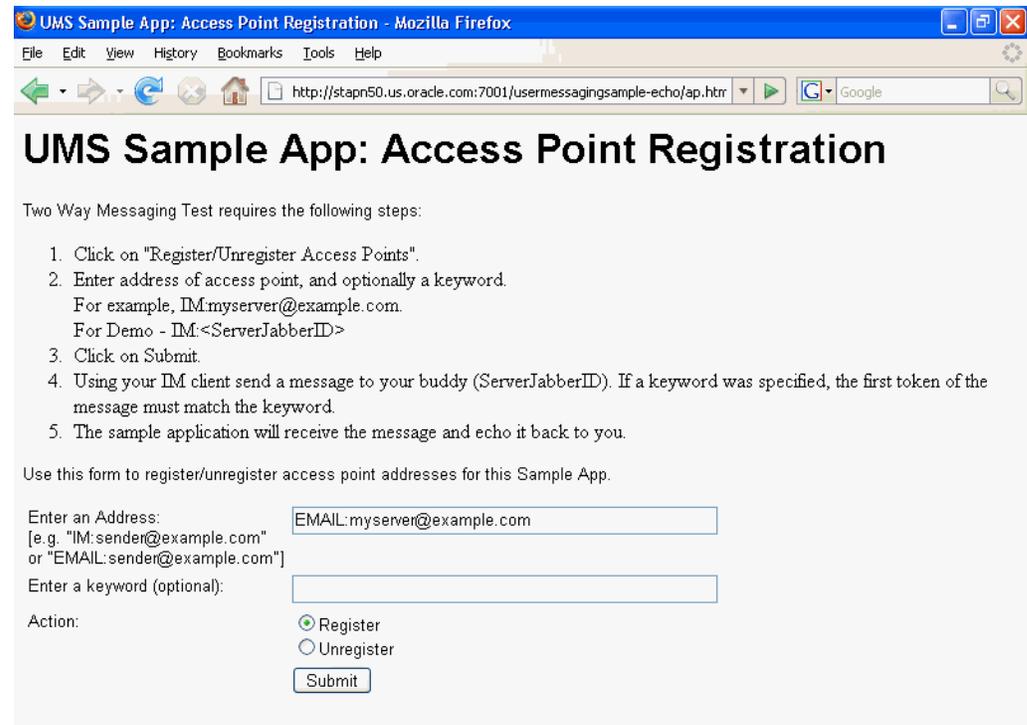
Once **usermessagingsample-echo** has been deployed to a running instance of Oracle WebLogic Server, perform the following:

1. Launch a web browser and enter the address of the sample application as follows: `http://host:http-port/usermessagingsample-echo/`. For example, enter `http://localhost:7001/usermessagingsample-echo/` into the browser's navigation bar.

When prompted, enter login credentials. For example, username `weblogic`. The browser page for testing messaging samples appears ([Figure 40–13](#)).

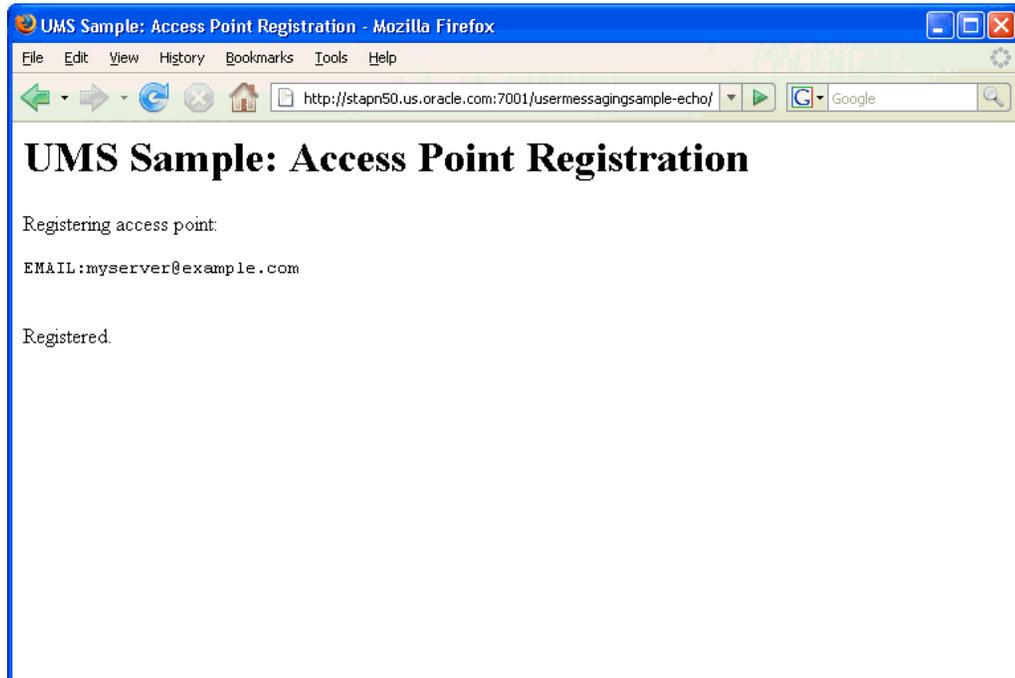
Figure 40–13 Testing the Sample Application

2. Click **Register/Unregister Access Points**. The *Access Point Registration* page appears (Figure 40–14).

Figure 40–14 Registering an Access Point

3. Enter the access point address in the following format:
`EMAIL:server_address.`
For example, enter `EMAIL:myserver@example.com`.
4. Select the Action **Register** and Click **Submit**. The registration status page appears, showing "Registered" in [Figure 40–15](#)).

Figure 40–15 Access Point Registration Status



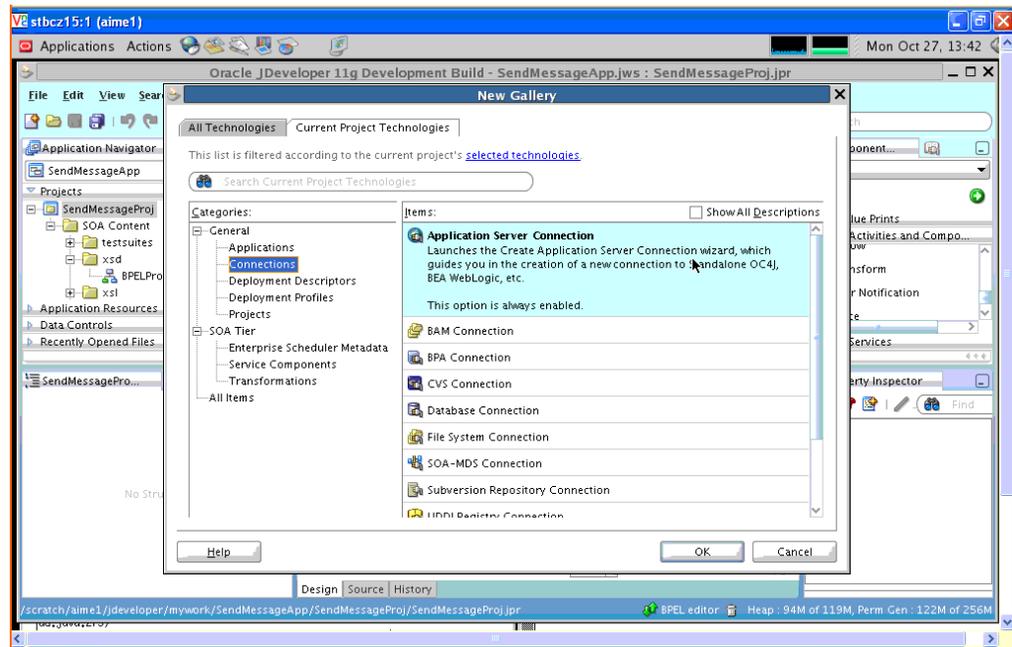
5. Send a message from your messaging client (for email, your email client) to the address you just registered as an access point in the previous step.
If the UMS messaging driver for that channel is configured correctly, you should expect to receive an echo message back from the **usermessagingsample-echo** application.

40.7 Creating a New Application Server Connection

Perform the following steps to create a new Application Server Connection.

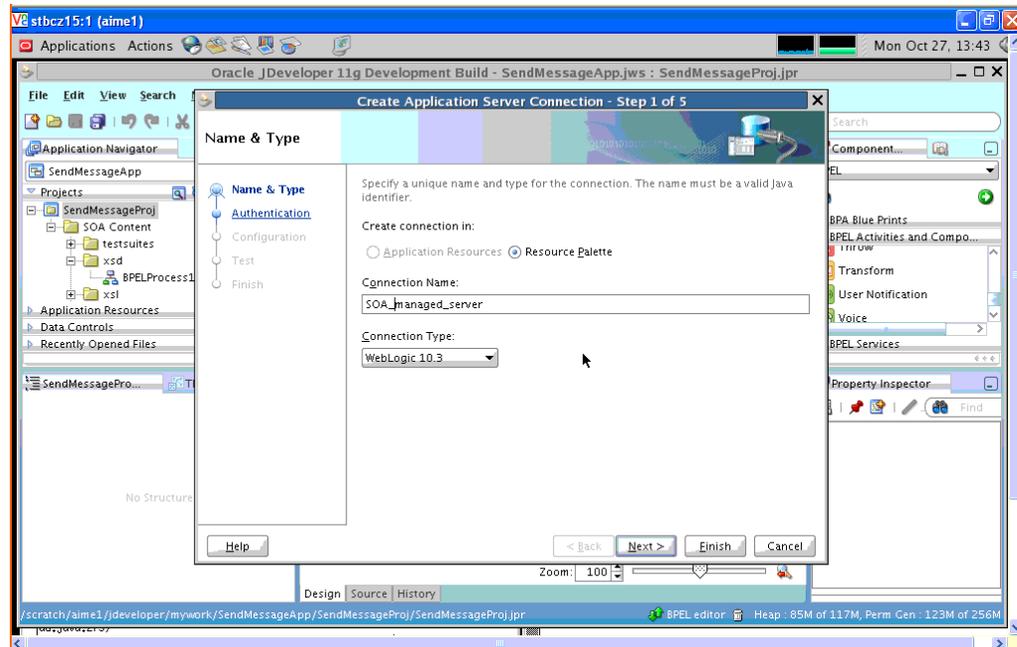
1. Create a new Application Server Connection by right-clicking the project and selecting **New, Connections, and Application Server Connection** ([Figure 40–16](#)).

Figure 40–16 New Application Server Connection



2. Name the connection SOA_server and click **Next** (Figure 40–17).
3. Select **WebLogic 10.3** as the **Connection Type**.

Figure 40–17 New Application Server Connection



4. Enter the authentication information. A typical value for user name is `weblogic`.
5. In the Connection dialog, enter the hostname, port, and SSL port for the SOA admin server, and enter the name of the domain for WLS Domain.
6. Click **Next**.

7. In the Test dialog, click **Test Connection**.
8. Verify that the message `Success!` appears.
The Application Server Connection has been created.

Parlay X Web Services Multimedia Messaging API

This chapter describes the Parlay X Multimedia Messaging Web Service that is available with Oracle User Messaging Service and how to use the Parlay X Web Services Multimedia Messaging API to send and receive messages through Oracle User Messaging Service.

Note: To learn about the architecture and components of Oracle User Messaging Service, see *Oracle Fusion Middleware Getting Started with Oracle SOA Suite*.

This chapter includes the following sections:

- [Section 41.1, "Overview of Parlay X Messaging Operations"](#)
- [Section 41.2, "Send Message Interface"](#)
- [Section 41.3, "Receive Message Interface"](#)
- [Section 41.4, "Oracle Extension to Parlay X Messaging"](#)
- [Section 41.5, "Parlay X Messaging Client API and Client Proxy Packages"](#)
- [Section 41.6, "Sample Chat Application with Parlay X APIs"](#)

Note: Oracle User Messaging Service also ships with a Java client library that implements the Parlay X API.

41.1 Overview of Parlay X Messaging Operations

The following sections describe the semantics of each of the supported operations along with implementation-specific details for the Parlay X Gateway. The following tables, describing input/output message parameters for each operation, are taken directly from the Parlay X specification.

Oracle User Messaging Service implements a subset of the Parlay X 2.1 Multimedia Messaging specification. Specifically Oracle User Messaging Service supports the `SendMessage` and `ReceiveMessage` interfaces. The `MessageNotification` and `MessageNotificationManager` interfaces are not supported.

41.2 Send Message Interface

The `SendMessage` interface allows you to send a message to one or more recipient addresses by using the `sendMessage` operation, or get the delivery status for a previously sent message by using the `getMessageDeliveryStatus` operation. The following requirements apply:

- A recipient address must conform to the address format requirements of Oracle User Messaging Service (in addition to being a valid URI). The general format is *delivery_type:protocol_specific_address*, such as `email:user@domain`, `sms:5551212` or `im:user@jabberdomain`.
- Certain characters are not allowed in URIs; if it is necessary to include them in an address they can be encoded or escaped. Refer to the JavaDoc for `java.net.URI` for details on how to create a properly encoded URI.
- While the WSDL specifies that sender addresses can be any string, Oracle User Messaging Service requires that they be valid Messaging addresses.
- Oracle User Messaging Service requires that you specify sender addresses on a per-delivery type basis. So for a sender address to apply to a recipient of a given delivery type, say EMAIL, the sender address must also have delivery type of EMAIL. Since this operation allows multiple recipient addresses but only one sender address, the sender address will only apply to the recipients with the same delivery type.
- Oracle User Messaging Service does not support the `MessageNotification` interface, and therefore will not produce delivery receipts, even if a `receiptRequest` is specified. In other words, the `receiptRequest` parameter is ignored.

41.2.1 sendMessage Operation

Table 41–1 describes message descriptions for the `sendMessageRequest` input in the `sendMessage` operation.

Table 41–1 *sendMessage Input Message Descriptions*

Part Name	Part Type	Optional	Description
addresses	xsd:anyURI[0..unbounded]	No	Destination address for this Message.
senderAddress	xsd:string	Yes	Message sender address. This parameter is not allowed for all 3rd party providers. The Parlay X server needs to handle this according to a SLA for the specific application and its use can therefore result in a <code>PolicyException</code> .
subject	xsd:string	Yes	Message subject. If mapped to SMS this parameter will be used as the senderAddress, even if a separate senderAddress is provided.
priority	MessagePriority	Yes	Priority of the message. If not present, the network will assign a priority based on the operator policy.Charging to apply to this message.

Table 41–1 (Cont.) *sendMessage* Input Message Descriptions

Part Name	Part Type	Optional	Description
charging	common:ChargingInformation	Yes	Charging to apply to this message.
receiptRequest	common:SimpleReference	Yes	Defines the application endpoint, interface name and correlator that will be used to notify the application when the message has been delivered to a terminal or if delivery is impossible.

Table 41–2 describes `sendMessageResponse` output messages for the `sendMessage` operation.

Table 41–2 *sendMessageResponse* Output Message Descriptions

Part Name	Part Type	Optional	Description
result	xsd:string	No	This correlation identifier is used in a <code>getMessageDeliveryStatus</code> operation invocation to poll for the delivery status of all sent messages.

41.2.2 `getMessageDeliveryStatus` Operation

The `getMessageDeliveryStatus` operation gets the delivery status for a previously sent message. The input "requestIdentifier" is the "result" value from a `sendMessage` operation. This is the same identifier that is referred to as a Message ID in other Messaging documentation.

Table 41–3 describes the `getMessageDeliveryStatusRequest` input messages for the `getMessageDeliveryStatus` operation.

Table 41–3 *getMessageDeliveryStatusRequest* Input Message Descriptions

Part Name	Part Type	Optional	Description
registrationIdentifier	xsd:string	No	Identifier related to the delivery status request.

Table 41–4 describes the `getMessageDeliveryStatusResponse` output messages for the `getMessageDeliveryStatus` operation.

Table 41–4 *getMessageDeliveryStatusResponse* Output Message Descriptions

Part Name	Part Type	Optional	Description
result	DeliveryInformation [0..unbounded]	Yes	An array of status of the messages that were previously sent. Each array element represents a sent message, its destination address and its delivery status.

41.3 Receive Message Interface

The `ReceiveMessage` interface has three operations. The `getReceivedMessages` operation polls the server for any messages received since the last invocation of `getReceivedMessages`. Note that `getReceivedMessages` does not necessarily return any message content; it generally only returns message metadata.

The other two operations, `getMessage` and `getMessageURIs`, are used to retrieve message content.

41.3.1 `getReceivedMessages` Operation

This operation polls the server for any received messages. Note the following requirements:

- The registration ID parameter is a string that identifies the endpoint address for which the application wants to receive messages. See the discussion of the `ReceiveMessageManager` interface for more details.
- The Parlay X specification says that if the registration ID is not specified, all messages for this application should be returned. However, the WSDL says that the registration ID parameter is mandatory. Therefore our implementation treats the empty string (""), as the "not-specified" value. If you call `getReceivedMessages` with the empty string as your registration ID, you will get all messages for this application. Therefore the empty string is not an allowed value of registration ID when calling `startReceiveMessages`.
- According to the Parlay X specification, if the received message content is "pure ASCII text", then the message content is returned inline within the `MessageReference` object, and the `messageIdentifier` (Message ID) element is null. Our implementation treats any content with Content-Type "text/plain", and with encoding "us-ascii" as "pure ASCII text" for the purposes of this operation. As per the MIME specification, if no encoding is specified, "us-ascii" is assumed, and if no Content-Type is specified, "text/plain" is assumed.
- The priority parameter is currently ignored.

[Table 41-5](#) describes the `getReceivedMessagesRequest` input messages for the `getReceivedMessages` operation.

Table 41-5 *getReceivedMessagesRequest Input Message Descriptions*

Part Name	Part Type	Optional	Description
registrationIdentifier	xsd:string	No	Identifies the off-line provisioning step that enables the application to receive notification of Message reception according to the specified criteria.
priority	MessagePriority	Yes	The priority of the messages to poll from the Parlay X gateway. All messages of the specified priority and higher will be retrieved. If not specified, all messages shall be returned, i.e. the same as specifying "Low."

[Table 41-6](#) describes the `getReceivedMessagesResponse` output messages for the `getReceivedMessages` operation.

Table 41–6 *getReceivedMessagesResponse Output Message Descriptions*

Part Name	Part Type	Optional	Description
registrationIdentifier	xsd:string	No	Identifies the off-line provisioning step that enables the application to receive notification of Message reception according to the specified criteria.
priority	MessagePriority	Yes	The priority of the messages to poll from the Parlay X gateway. All messages of the specified priority and higher will be retrieved. If not specified, all messages shall be returned. This is the same as specifying Low.

41.3.2 getMessage Operation

The `getMessage` operation retrieves message content, using a message ID from a previous invocation of `getReceivedMessages`. There is no SOAP body in the response message; the content is returned as a single SOAP attachment.

Table 41–7 describes the `getMessageRequest` input messages for the `getMessage` operation.

Table 41–7 *getMessageRequest Input Message Descriptions*

Part Name	Part Type	Optional	Description
messageRefIdentifier	xsd:string	No	The identity of the message.

There are no `getMessageResponse` output messages for the `getMessage` operation.

41.3.3 getMessageURIs Operation

The `getMessageURIs` retrieves message content as a list of URIs. Note the following requirements:

- These URIs will be HTTP URLs which can be dereferenced to retrieve the content.
- If the inbound message has a Content-Type of "multipart", then there will be multiple URIs returned, one per sub-part. If the Content-Type is not "multipart", then a single URI will be returned.
- Per the Parlay X specification, if the inbound messages a body text part, defined as "the message body if it is encoded as ASCII text", it is returned inline within the MessageURI object. For the purposes of our implementation, we define this behavior as follows:
 - If the message's Content-Type is "text/*" (any text type), and if the charset parameter is "us-ascii", then the content is returned inline in the MessageURI object. There will be no URI returned since there is no content other than what is returned inline.
 - If the message's Content-Type is "multipart/" (any multipart type), and if the first body part's Content-Type is "text/" with charset "us-ascii", then that part is returned inline in the MessageURI object, and there will be no URI returned corresponding to that part.

- Per the MIME specification, if the charset parameter is omitted, the default value of "us-ascii" is assumed. If the Content-Type header is not specified for the message, then a Content-Type of "text/plain" is assumed.

Table 41–8 describes the `getMessageURIsRequest` input messages for the `getMessageURIs` operation.

Table 41–8 *getMessageURIsRequest Input Message Descriptions*

Part Name	Part Type	Optional	Description
messageRefIdentifier	xsd:string	No	The identity of the message to retrieve.

Table 41–9 describes the `getMessageURIsResponse` output messages for the `getMessageURIs` operation.

Table 41–9 *getMessageURIsResponse Output Message Descriptions*

Part Name	Part Type	Optional	Description
result	MessageURI	No	Contains the complete message, consisting of the textual part of the message, if such exists, and a list of file references for the message attachments, if any.

41.4 Oracle Extension to Parlay X Messaging

The Parlay X Messaging specification leaves certain parts of the messaging flow undefined. The main area that is left undefined is the process for binding a client to an address for synchronous receiving (through the `ReceiveMessage` interface).

Oracle User Messaging Service includes an extension interface to Parlay X to support this process. The extension is implemented as a separate WSDL in an Oracle XML namespace to indicate that it is not an official part of Parlay X. Clients can choose to not use this additional interface or use it in some modular way such that their core messaging logic remains fully compliant with the Parlay X specification.

41.4.1 ReceiveMessageManager Interface

`ReceiveMessageManager` is the Oracle-specific interface for managing client registrations for receiving messages. Clients use this interface to start and stop receiving messages at a particular address. (This is analogous to the concept of registering/unregistering access points in the Messaging API).

41.4.1.1 startReceiveMessage Operation

Invoking this operation allows a client to bind itself to a given endpoint for the purpose of receiving messages. Note the following requirements:

- An endpoint consists of an address and an optional "criteria", defined by the Parlay X specification as the first white space-delimited token of the message subject or content.
- In addition to the endpoint information, the client also specifies a "registration ID" when invoking this operation; this ID is just a unique string which can be used later to refer to this particular binding in the `stopReceiveMessage` and `getReceivedMessages` operations.

- If an endpoint is already registered by another client application, or the registration ID is already being used, a Policy Error will result.
- Certain characters are not allowed in URIs; if it is necessary to include them in an address they can be encoded/escaped. See the javadoc for `java.net.URI` for details on how to create a properly encoded URI. For example, when registering to receive XMPP messages you must specify an address such as `IM:jabber|user@example.com`, however the pipe (`|`) character is not allowed in URIs, and must be escaped before submitting to the server.
- There is no guarantee that the server can actually receive messages at a given endpoint address. That depends on the overall configuration of Oracle User Messaging Service, particularly the Messaging drivers that are deployed in the system. No error will be indicated if a client binds to an address where the server cannot receive messages.

The `startReceiveMessage` operation has the following inputs and outputs:

[Table 41-10](#) describes the `startReceiveMessageRequest` input messages for the `startReceiveMessage` operation.

Table 41-10 *startReceiveMessageRequest Input Message Descriptions*

Part Name	Part Type	Optional	Description
registrationIdentifier	xsd:string	No	A registration identifier.
messageServiceActivationNumber	xsd:anyURI	No	Message Service Activation Number.
criteria	xsd:string	Yes	Descriptive string.

There are no `startReceiveMessageResponse` output messages for the `startReceiveMessage` operation.

41.4.1.2 stopReceiveMessage Operation

Invoking this operation removes the previously-established binding between a client and a receiving endpoint. The client specifies the same registration ID that was supplied when `startReceiveMessage` was called in order to identify the endpoint binding that is being broken. If there is no corresponding registration ID binding known to the server for this application, a Policy Error will result.

[Table 41-11](#) describes the `stopReceiveMessageRequest` input messages for the `stopReceiveMessage` operation.

Table 41-11 *stopReceiveMessageRequest Input Message Descriptions*

Part Name	Part Type	Optional	Description
registrationIdentifier	xsd:string	No	A registration identifier.

There are no `stopReceiveMessageResponse` output messages for the `stopReceiveMessage` operation.

41.5 Parlay X Messaging Client API and Client Proxy Packages

While it is possible to assemble a Parlay X Messaging Client using only the Parlay X WSDL files and a web service assembly tool, we also provide pre-built web service stubs and interfaces for the supported Parlay X Messaging interfaces. Due to difficulty

in assembling a web service with SOAP attachments in the style mandated by Parlay X, we recommend the use of the provided API rather than starting from WSDL.

For a complete listing of the classes available in the Parlay X Messaging API, see the Messaging JavaDoc. The main entry points for the API are through the following client classes:

- `oracle.sdp.parlayx.multimedia_messaging.send.SendMessageClient`
- `oracle.sdp.parlayx.multimedia_messaging.receive.ReceiveMessageClient`
- `oracle.sdp.parlayx.multimedia_messaging.extension.receive_manager.ReceiveMessageManager`

Each client class allows a client application to invoke the operations in the corresponding interface. Additional web service parameters such as the remote gateway URL and any required security credentials, are provided when an instance of the client class is constructed. See the Javadoc for more details. The security credentials will be propagated to the server using standard WS-Security headers, as mandated by the Parlay X specification.

The general process for a client application is to create one of the client classes above, set the necessary configuration items (endpoint, username, password), then invoke one of the business methods (for example, `SendMessageClient.sendMessage()`, and so on). For examples of how to use this API, see the Messaging samples on Oracle Technology Network (OTN), and specifically `usermessagingsample-parlayx-src.zip`.

41.6 Sample Chat Application with Parlay X APIs

This chapter describes how to create, deploy and run the sample chat application with Parlay X APIs provided with Oracle User Messaging Service on OTN.

Note: To learn about the architecture and components of Oracle User Messaging Service, see *Oracle Fusion Middleware Getting Started with Oracle SOA Suite*.

This chapter contains the following sections:

- [Section 41.6.1, "Overview"](#)
- [Section 41.6.2, "Running the Pre-Built Sample"](#)
- [Section 41.6.3, "Testing the Sample"](#)
- [Section 41.6.4, "Creating a New Application Server Connection"](#)

41.6.1 Overview

This sample demonstrates how to create a web-based chat application to send and receive messages through email, SMS, or IM. The sample uses standards-based Parlay X Web Service APIs to interact with a User Messaging server. The sample application includes web service proxy code for each of three web service interfaces: the `SendMessage` and `ReceiveMessage` services defined by Parlay X, and the `ReceiveMessageManager` service which is an Oracle extension to Parlay X. You define an application server connection in Oracle JDeveloper, and deploy and run the application.

The application is provided as a pre-built Oracle JDeveloper project that includes a simple web chat interface.

41.6.1.1 Provided Files

The following files are included in the sample application:

- Project – the directory containing the archived Oracle JDeveloper project files.
- Readme.txt.
- Release notes

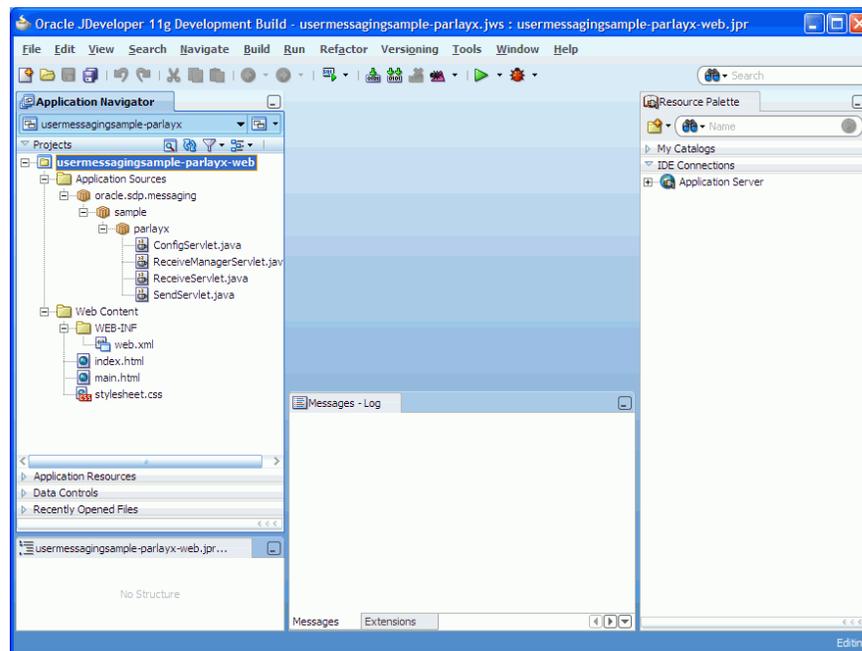
41.6.2 Running the Pre-Built Sample

Perform the following steps to run and deploy the pre-built sample application:

1. Open the **usermessagingsample-parlayx.jws** (contained in the .zip file) in Oracle JDeveloper.

In the Oracle JDeveloper main window the project appears.

Figure 41–1 Oracle JDeveloper Main Window

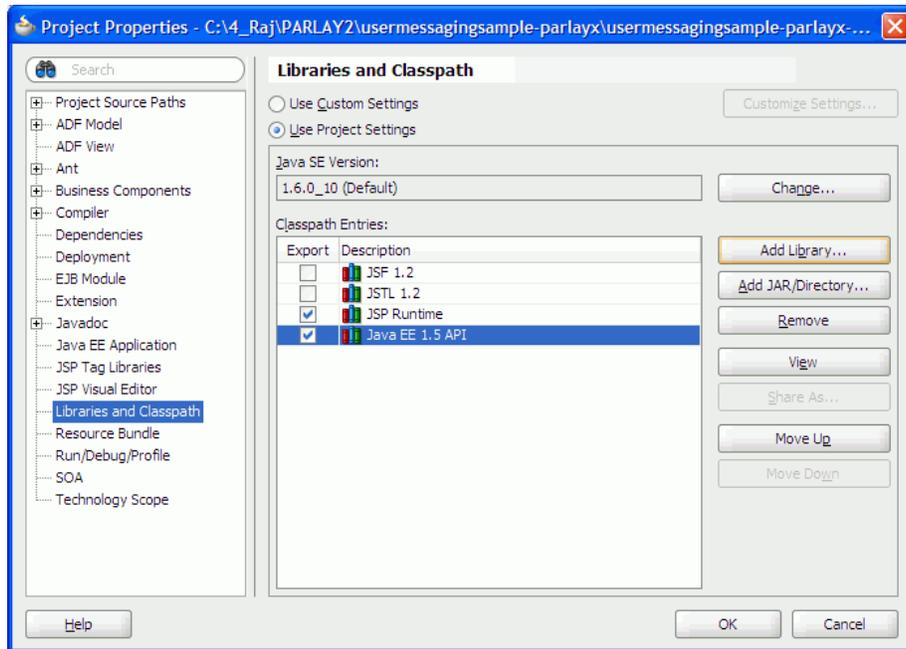


2. In Oracle JDeveloper, select **File > Open...**, then navigate to the directory above and open workspace file "usermessagingsample-parlayx.jws".

This will open the precreated JDeveloper application for the Parlay X sample application. The application contains one web module. All of the source code for the application is in place. You will need to configure the parameters that are specific to your installation.

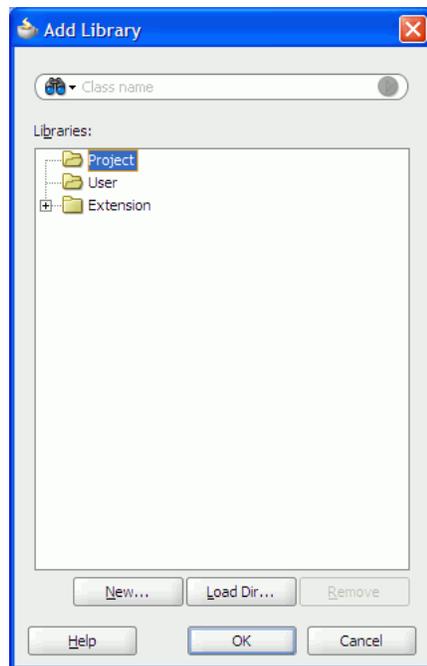
3. Satisfy the build dependencies for the sample application by adding a library to the web module.
 1. In the Application Navigator, right-click web module usermessagingsample-parlayx-war, and select **Project Properties**.
 2. In the left pane, select **Libraries and Classpath**.

Figure 41–2 Adding a Library



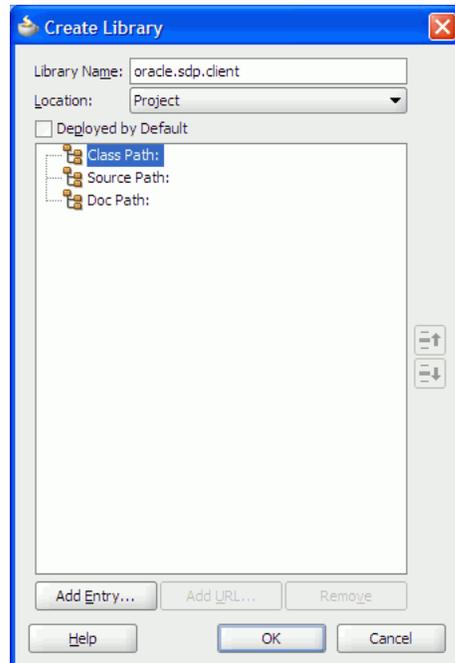
3. Click Add Library.

Figure 41–3 Adding a Library

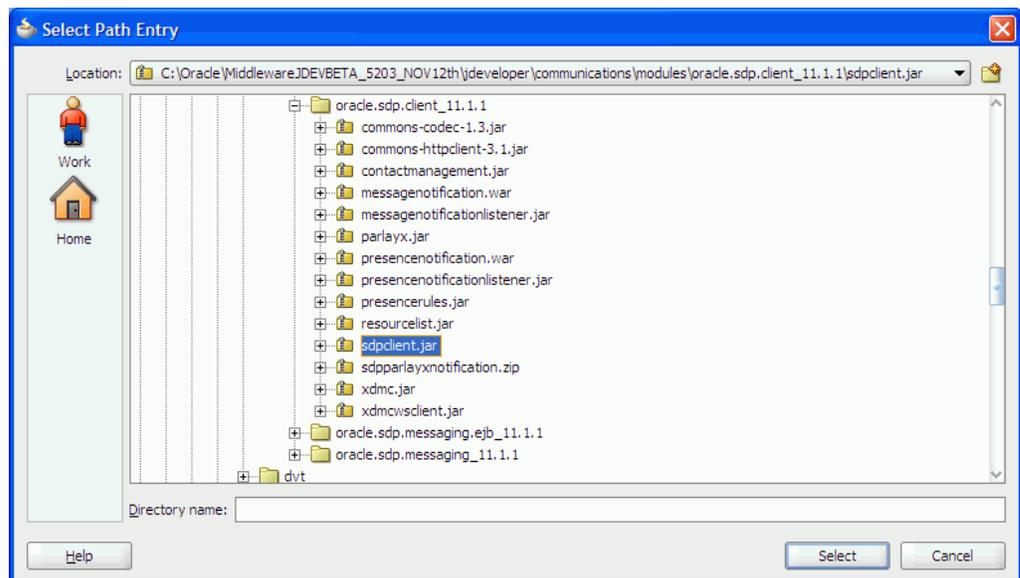


4. Click New to define a new library.

5. For Library Name, enter oracle.sdp.client.

Figure 41–4 Defining the Library

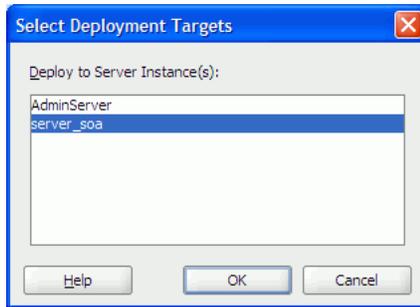
6. With **Class Path** selected, select **Add Entry**.
7. Navigate to ***JDeveloper_Base_Directory/communications/modules/oracle.sdp.client_11.1.1***, and select jar file ***sdpclient.jar***.

Figure 41–5 Selecting sdpclient.jar

8. Click **OK/Accept** in all popups to create the library and add it as a dependency to the sample web module.
4. Create an Application Server Connection by right-clicking the project in the navigation pane and selecting **New**. Follow the instructions in [Section 41.6.4, "Creating a New Application Server Connection"](#).

5. Deploy the project by selecting the **usermessasgingsample-parlayx** project, **Deploy**, **usermessasgingsample-parlayx**, **to**, and **SOA_server** (Figure 41–6).

Figure 41–6 Deploying the Project



6. Verify that the message `Build Successful` appears in the log.
7. Enter the default revision and click **OK**.
8. Verify that the message `Deployment Finished` appears in the deployment log.

You have successfully deployed the application.

Before you can run the sample you must configure any additional drivers in Oracle User Messaging Service and configure a default device for the user receiving the message in User Messaging Preferences, as described in the following sections.

Note: Refer to *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite* for more information.

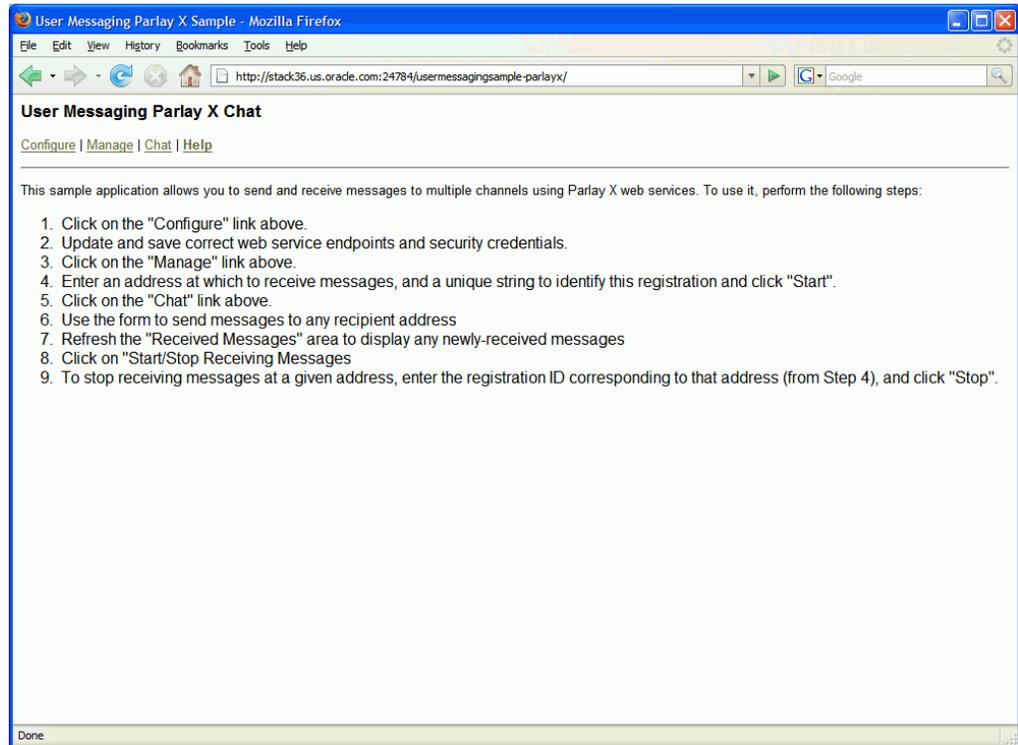
41.6.3 Testing the Sample

Perform the following steps to run and test the sample:

1. Open a web browser.
2. Navigate to the URL of the application as follows, and log in:

`http://host:port/usermessasgingsample-parlayx/`

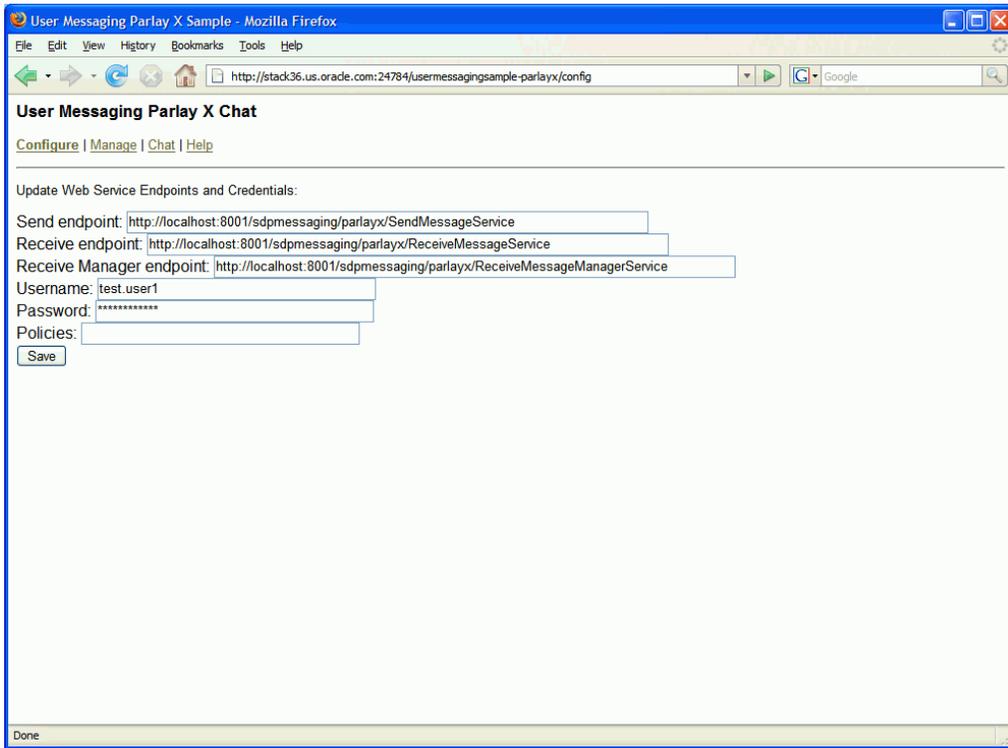
The Messaging Parlay X Sample web page appears (Figure 41–7). This page contains navigation tabs and instructions for the application.

Figure 41–7 Messaging Parlay X Sample Web Page

3. Click **Configure** and enter the following values (Figure 41–8):

- Specify the Send endpoint. For example, `http://localhost:port/sdpMessaging/parlayx/SendMessageService`
- Specify the Receive endpoint. For example, `http://localhost:port/sdpMessaging/parlayx/ReceiveMessageService`
- Specify the Receive Manager endpoint. For example, `http://localhost:port/sdpMessaging/parlayx/ReceiveMessageManagerService`
- Specify the Username and Password.
- Specify a Policy (required if the User Messaging Service instance has WS security enabled).

Figure 41–8 Configuring the Web Service Endpoints and Credentials



4. Click **Save**.
5. Click **Manage**.
6. Enter a Registration ID to specify the registration and address at which to receive messages (Figure 41–9). You can also use this page to stop receiving messages at an address.

Figure 41–9 Specifying a Registration ID

User Messaging Parlay X Chat

Configure | Manage | Chat | Help

Register an address at which to receive messages:

Registration ID: Address: Keyword:

Stop receiving on a previously registered address:

Registration ID:

Done

7. Click **Start**.

Verify that the message Registration operation succeeded appears.

8. Click **Chat** (Figure 41–10).

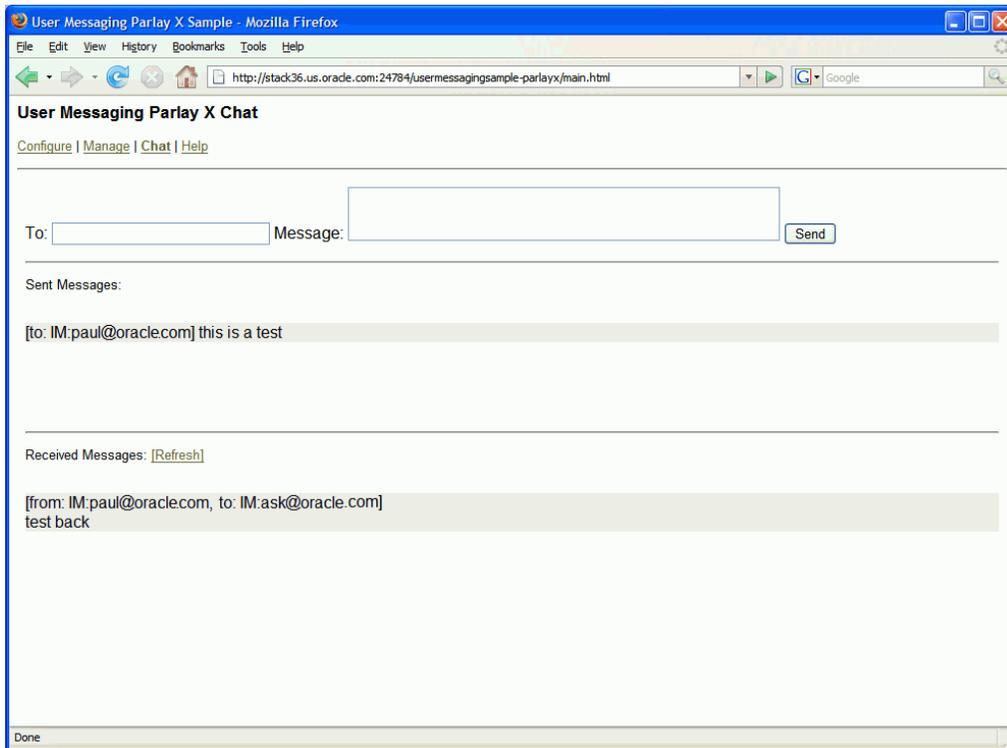
9. Enter recipients in the **To:** field in the format illustrated in Figure 41–10.

10. Enter a message.

11. Click **Send**.

12. Verify that the message is received.

Figure 41–10 Running the Sample

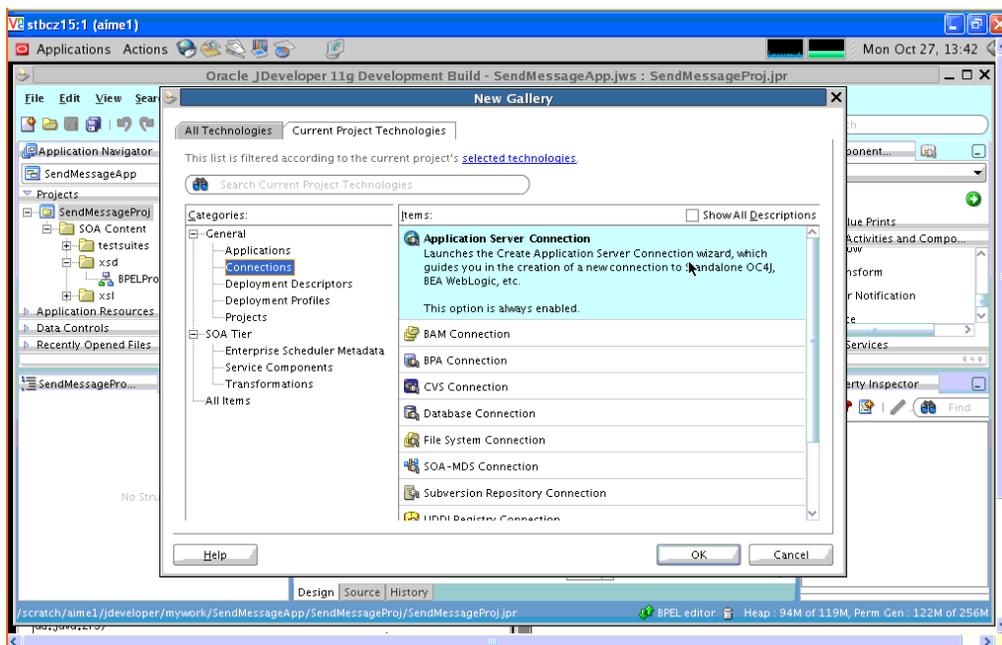


41.6.4 Creating a New Application Server Connection

Perform the following steps to create a new Application Server Connection.

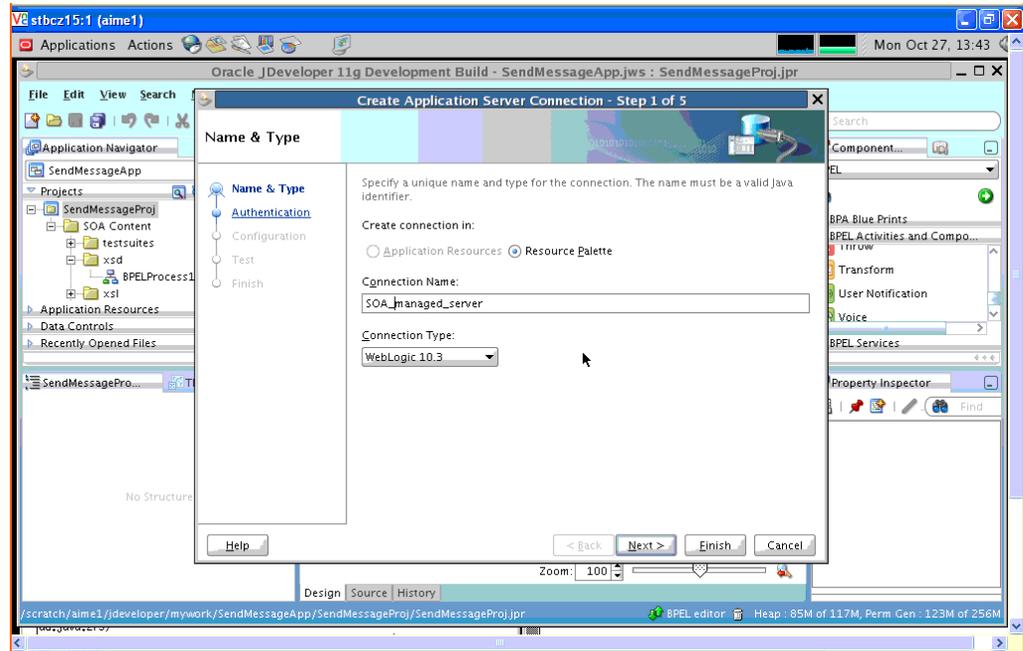
1. Create a new Application Server Connection by right-clicking the project and selecting **New, Connections, and Application Server Connection** (Figure 41–11).

Figure 41–11 New Application Server Connection



2. Name the connection SOA_server and click **Next** (Figure 41-12).
3. Select **WebLogic 10.3** as the **Connection Type**.

Figure 41-12 New Application Server Connection



4. Enter the authentication information. The typical value for username is `weblogic`.
5. In the Connection dialog, enter the hostname, port and SSL port for the SOA admin server, and enter the name of the domain for WLS Domain.
6. Click **Next**.
7. On the Test dialog, click **Test Connection**.
8. Verify that the message `Success!` appears.
The Application Server Connection has been created.

User Messaging Preferences

This chapter describes the User Messaging Preferences that are packaged with Oracle User Messaging Service. It describes how to work with messaging channels and to create contact rules using messaging filters.

Note: To learn about the architecture and components of Oracle User Messaging Service, see *Oracle Fusion Middleware Getting Started with Oracle SOA Suite*.

This chapter contains the following sections:

- [Section 42.1, "Introduction"](#)
- [Section 42.2, "How to Manage Messaging Channels"](#)
- [Section 42.3, "Creating Contact Rules using Filters"](#)
- [Section 42.4, "Configuring Settings"](#)

42.1 Introduction

User Messaging Preferences allows a user who has access to multiple channels (delivery types) to control how, when, and where they receive messages. Users define filters, or delivery preferences, that specify which channel a message should be delivered to, and under what circumstances. Information about a user's devices and filters are stored in any database supported for use with Oracle Fusion Middleware.

For an application developer, User Messaging Preferences provide increased flexibility. Rather than an application needing business logic to decide whether to send an email or SMS message, the application can just send to the user, and the message will be delivered according to the user's preferences.

Since preferences are stored in a database, this information is shared across all instances of User Messaging Preferences in a domain.

The `oracle.sdp.messaging.userprefs` package contains the User Messaging Preferences API classes. For more information, refer to the Javadoc.

42.1.1 Terminology

User Messaging Preferences defines the following terminology:

- Channel: a physical channel, such as a phone, or PDA.
- Channel address: one of the addresses that a channel can communicate with.

- Filters: a set of notification delivery preferences.
- System term: a pre-defined business term that cannot be extended by the administrator.
- Business term: a rule term defined and managed by the system administrator through Enterprise Manager. Business terms can be added, defined, or deleted.
- Rule term: a system term or a business term.
- Operators: comparison operators *equals*, *does not equal*, *contains*, or *does not contain*.
- Facts: data passed in from the message to be evaluated, such as *time sent*, or *sender*.
- Rules Engine: the User Messaging Preferences component that processes and evaluates filters.
- Channel: the transport type, for example, email, voice, or SMS.
- Comparison: a rule term and the associated comparison operator.
- Action: the action to be taken if the specified conditions in a rule are true, such as *Broadcast to All*, *Failover*, or *Do not Send to Any Channel*.

42.1.2 Configuration of Notification Delivery Preferences

User Messaging Preferences allows configuration of notification delivery preferences based on the following:

- a set of well-defined rule terms (system terms or business terms)
- a set of channel and the corresponding addresses supported by Oracle User Messaging Service
- a set of User Messaging Preferences filters that are transparently handled by a rules engine

One use case for notification delivery preference is for bugs entered into a bug tracking system. For example, user *Alex* wants to be notified through SMS and EMAIL channels for bugs filed against his product with priority = 1 by a customer type = Premium. For all other bugs with priority > 1, he only wants to be notified by EMAIL. Alex's preferences can be stated as follows:

Example 42-1 Notification Delivery Preferences

Rule (1): if (Customer Type = Premium) AND (priority = 1) then notify [Alex] using SMS and EMAIL.

Rule (2): if (Customer Type = Premium) AND (priority > 1) then notify [Alex] using EMAIL.

A runtime service, the Oracle Rules Engine, evaluates the filters to process the notification delivery of user requests.

42.1.3 Delivery Preference Rules

A delivery preference rule consists of *rule comparisons* and *rule actions*. A rule comparison consists of a *rule term* (a system term or a business term) and the associated comparison operators. A rule action is the action to be taken if the specified conditions in a rule are true.

42.1.3.1 Data Types

Table 42–2 lists data types supported by User Messaging Preferences. Each system term and business term must have an associated data type, and each data type has a set of pre-defined comparison operators. Administrators cannot extend these operators.

Table 42–1 Data Types Supported by User Messaging Preferences

Data Type	Comparison Operators	Supported Values
Date	<, >, between, <=, >=	Date is accepted as a <code>java.util.Date</code> object or string representing the number of milliseconds since the standard base time known as "the epoch", namely January 1, 1970, 00:00:00 GMT (in essence, the value from <code>java.util.Date.getTime()</code> or <code>java.util.Calendar.getTime()</code>).
Time	==, !=, between	A 4-digit integer to represent time of the day in HHMM format. First 2-digit is the hour in 24-hour format. Last 2-digit is minutes.
Number (Decimal)	<, >, between, <=, >=	A <code>java.lang.Double</code> object or a string representing a floating decimal point number with double precision.
String	==, !=, contains, not contains	Any arbitrary string.

Note: The String data type does not support regular expressions.
The Time data type is only available to System Terms.

42.1.3.2 System Terms

Table 42–2 lists system terms, which are pre-defined business terms. Administrators cannot extend the system terms.

Table 42–2 System Terms Supported by User Messaging Preferences

System Term	Data Type	Supported Values
Date	Date	Date is accepted as a <code>java.util.Date</code> object or string representing the number of milliseconds since the standard base time known as "the epoch", namely January 1, 1970, 00:00:00 GMT (in essence, the value from <code>java.util.Date.getTime()</code> or <code>java.util.Calendar.getTime()</code>).
Time	Time	A 4-digit integer to represent time of the day in HHMM format. First 2-digit is the hour in 24-hour format. Last 2-digit is minutes.

42.1.3.3 Business Terms

Business terms are rule terms defined and managed by the system administrator through Oracle Application Server 11g Enterprise Manager. For more information on adding, defining, and deleting business terms, refer to *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite*. A business term consists of a key, a data type, an optional description, and an optional List of Values (LOV).

Table 42–3 lists the pre-defined business terms supported by User Messaging Preferences.

Table 42–3 Pre-defined Business Terms for User Messaging Preferences

Business Term	Data Type
Organization	String
Time	Number (Decimal)
Priority	String
Application	String
Application Type	String
Expiration Date	Date
From	String
To	String
Customer Name	String
Customer Type	String
Status	String
Amount	Number (Decimal)
Due Date	Date
Process Type	String
Expense Type	String
Total Cost	Number (Decimal)
Processing Time	Number (Decimal)
Order Type	String
Service Request Type	String
Group Name	String
Source	String
Classification	String
Duration	Number (Decimal)
User	String
Role	String

42.1.4 Rule Actions

For a given rule, a User Messaging Preferences user can define one of the following actions:

- **Broadcast to All:** send a broadcast message to all channels in the broadcast address list.
- **Failover:** Send a message serially to channels in the address list until one successful message is sent. This means performing a send to the next channel when the current channel returns a failure status. User Messaging Preferences does not allow a user to specify a channel-specific status code or expiration time.
- **Do not send to Any Channel:** Do not send a message to any channel.

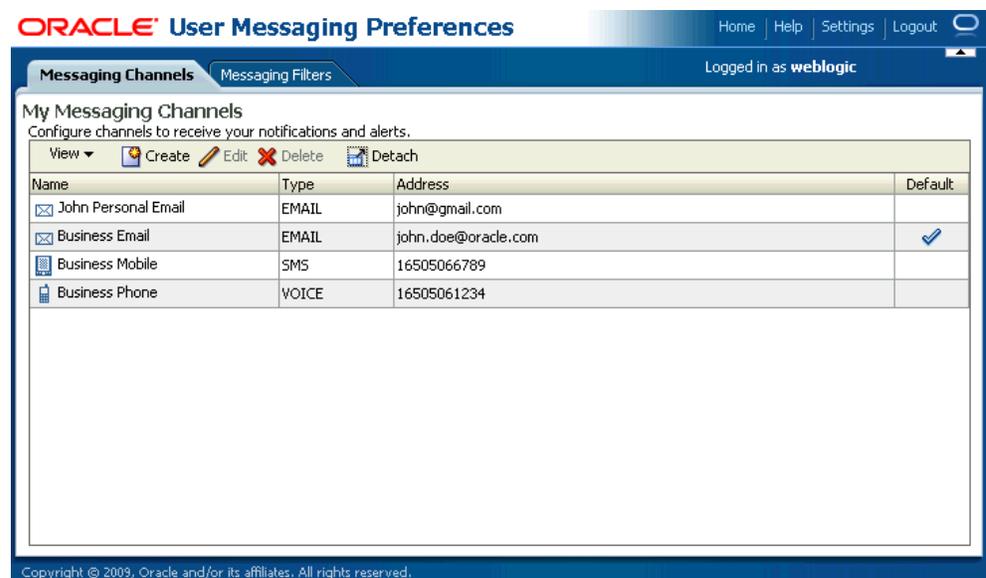
Tip: User Messaging Preferences does not provide a filter action that instructs "do not send to a specified channel." A best practice is to specify only positive actions, and not negative actions in rules.

- **Default address:** if no action is defined, a message will be sent to a default address, as defined in the Messaging Channels page in Enterprise Manager.

42.2 How to Manage Messaging Channels

Any channel that a user creates is associated with that user's system ID. In Oracle User Messaging Service, channels represent both physical channels, such as mobile phones, and also email client applications running on desktops, and are configurable on the *The Messaging Channels* tab (Figure 42-1).

Figure 42-1 *Messaging Channels Tab*



The *Messaging Channels* tab enables users to perform the following tasks:

42.2.1 Creating a Channel

To create a channel:

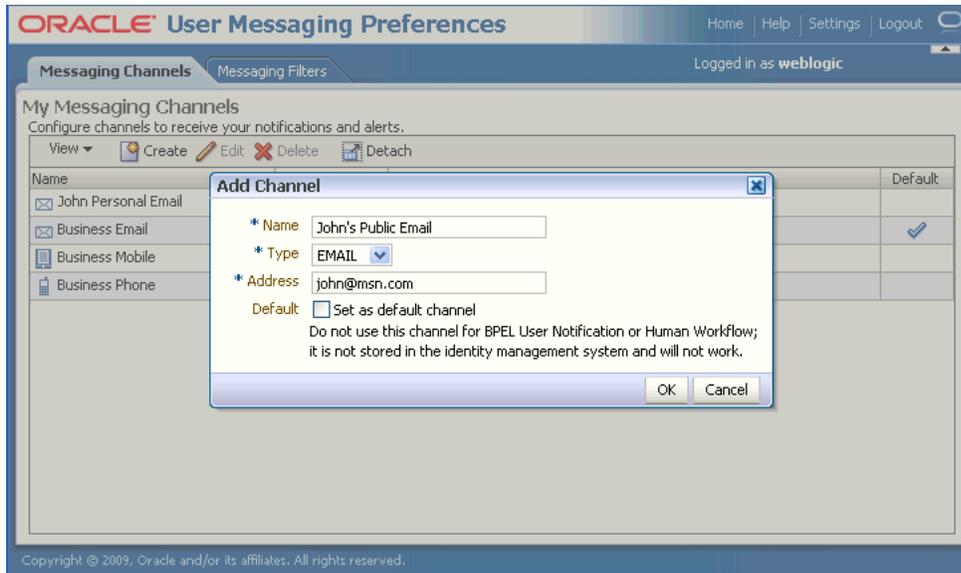
1. Click **Create** (Figure 42-2).

Figure 42-2 *The Create Icon*



2. Enter a name for the channel in the **Name** field (Figure 42-3).
3. Select the channel's transport type from the **Type** dropdown menu.
4. Enter the number or address appropriate to the transport type you selected.
5. Select the **Default** checkbox to set the channel as the default channel.

Figure 42–3 Creating a Channel

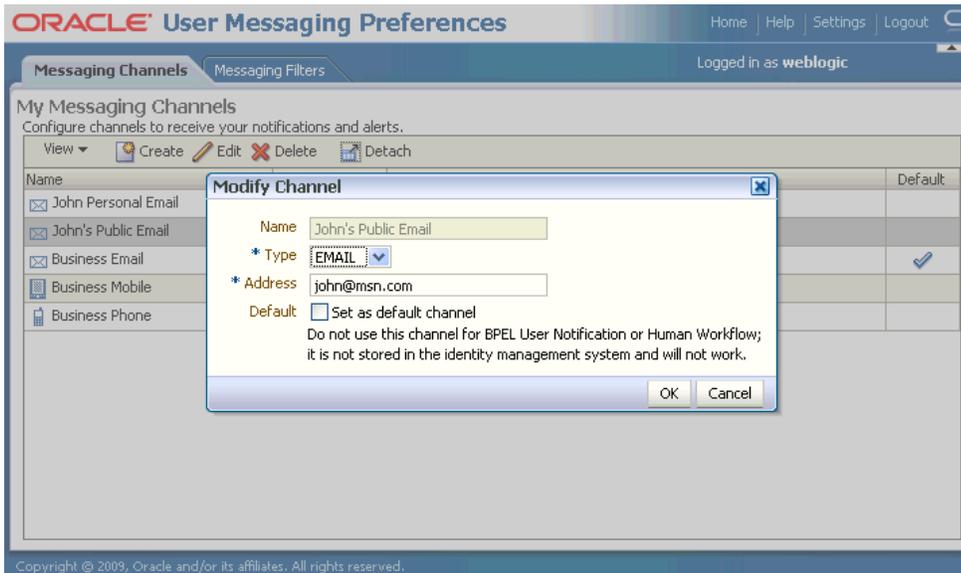


6. Click **OK** to create the channel. The channel appears on the *Channels* page. The *Channels* page enables you to edit or delete the channel.

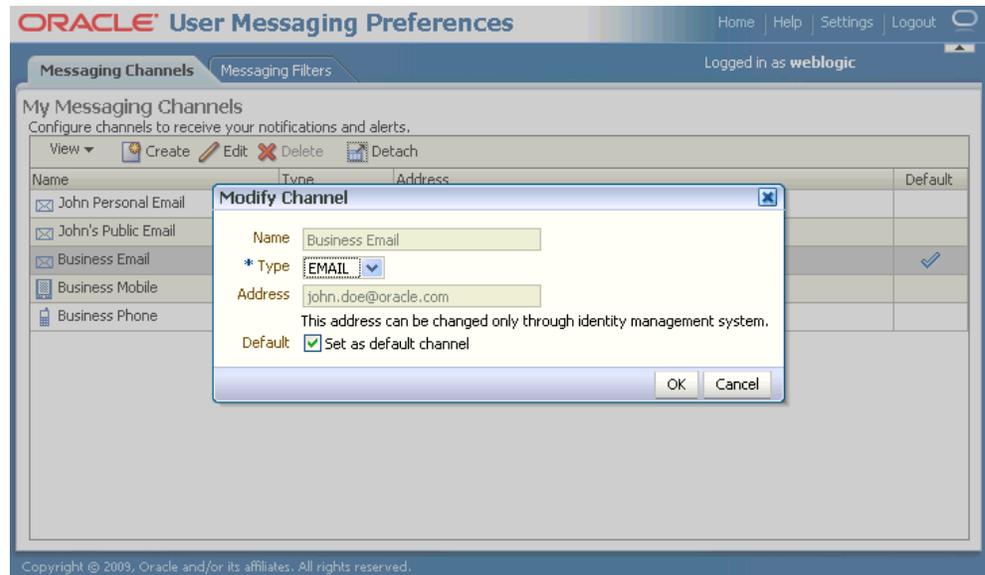
42.2.2 Editing a Channel

To edit a channel, select it and click **Edit** (Figure 42–4). The editing page appears for the channel, which enables you to add or change the channel properties described in Section 42.2.1, "Creating a Channel".

Figure 42–4 Edit a Channel



Certain channels are based on information retrieved from your user profile in the identity store, and this address cannot be modified by User Messaging Preferences (Figure 42–5). The only operation that can be performed on such as channel is to make it the default.

Figure 42–5 Edit a Identity Store-Backed Channel

42.2.3 Deleting a Channel

To delete a channel, select it and click **Delete** (Figure 42–6).

Figure 42–6 The Delete Icon

42.2.4 Setting a Default Channel

Email is the default for receiving notifications. To set another channel as the default, select it, click **Edit**, and then click **Set as default channel**. A check mark (Figure 42–7) appears next to the selected channel, designating it as the default means of receiving notifications.

Figure 42–7 The Default Icon

42.3 Creating Contact Rules using Filters

The **Messaging Filters** tab (Figure 42–8) enables users to build filters that specify not only the type of notifications they wish to receive, but also the channel through which to receive these notifications through a combination of comparison operators (such as *is equal to*, *is not equal to*), business terms that describe the notification type, content or source, and finally, the notification actions, which send the notifications to all channels, block channels from receiving notifications, or send notifications to the first available channel.

Figure 42–8 Messaging Filters Tab

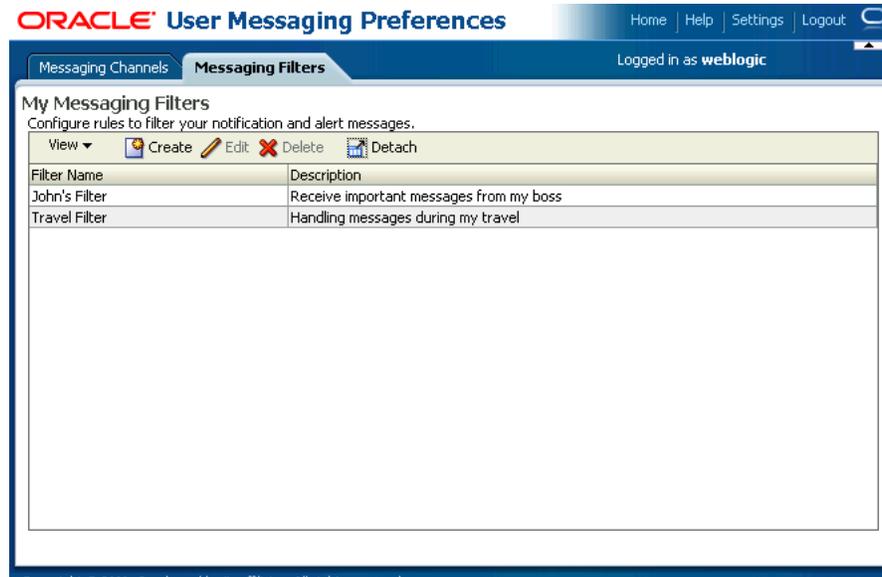


Figure 42–9 illustrates the creation of a filter called Travel Filter, by a user named weblogic, for handling notifications regarding Customers during his travel. Notifications that match all of the filter conditions are first directed to his "Business Mobile" channel. Should this channel become unavailable, Oracle User Messaging Service transmits the notifications as e-mails since the next available channel selected is Business Email.

Figure 42–9 Creating a Filter

ORACLE User Messaging Preferences

Home | Help | Settings | Logout

Logged in as weblogic

Messaging Channels | **Messaging Filters**

Your Reference System: Wednesday, January 14, 2009 10:30:12 AM
 Time: PST

* Filter Name: Travel Filter
 Description: Handling messages during my travel

Condition

Matching: All of the following conditions

Add Filter Condition: Subject Contains * Customer

Attribute	Operator	Value	Value2 (if required)	Delete
Date	Between	08/08/2008	10/28/2008	✗
Subject	Contains	Customer		✗

Action

Messaging Option: Send to the First Available Channel

Add Notification Channel:

Channel	Address	Up	Down	Delete
Business Mobile	16505066789	↑	↓	✗
Business Email	john.doe@oracle.com	↑	↓	✗

Copyright © 2009, Oracle and/or its affiliates. All rights reserved.

42.3.1 Creating Filters

To create a filter:

1. Click **Create** (Figure 42–2). The Create Filter page appears (Figure 42–9).
2. Enter a name for the filter in the **Filter Name** field.
3. If needed, enter a description of the filter in the **Description** field.
4. Define the filter conditions using the lists and fields of the **Condition** section as follows:
 - a. Select whether notifications must meet all of the conditions or any of the conditions by selecting either the **All of the following conditions** or the **Any of the following conditions** options.
 - b. Select the notification's attributes. These attributes, or business components, include
 - Organization
 - Time
 - Priority
 - Application
 - Application Type
 - Expiration Date

- From
 - To
 - Customer Name
 - Customer Type
 - Status
 - Amount
 - Due Date
 - Process Type
 - Expense Type
 - Total Cost
 - Processing Time
 - Order Type
 - Service Request Type
 - Group Name
 - Source
 - Classification
 - Duration
 - User
 - Role
5. Combine the selected condition type with one of the following comparison operators:
- Is Equal To
 - Is Not Equal To
 - Contains
 - Does Not Contain

If you select the **Date** attribute, select one of the following comparison operators and then select the appropriate dates from the calendar application.

- Is Equal
 - Is Not Equal
 - Is Greater Than
 - Is Greater Than or Equal
 - Is Less Than
 - Is Less Than or Equal
 - Between
 - Is Weekday
 - Is Weekend
6. Add appropriate values describing the attributes or operators.

7. Click **Add** (Figure 42–6) to add the attribute and the comparison operators to the table.
8. Repeat these steps to add more filter conditions. To delete a filter condition, click **Delete** (Figure 42–6).
9. Select one of the following delivery rules:
 - **Send Messages to all Selected Channels** -- Select this option to send messages to every listed channel.
 - **Send to the First Available Channel (Failover in the order)** -- Select this option to send messages matching the filter criteria to a preferred channel (set using the up and down arrows) or to the next available channel.
 - **Send No Messages** -- Select this option to block the receipt of any messages that meet the filter conditions.
10. To set the delivery channels, select a channel from the **Add Notification Channel** list and then click **Add** (Figure 42–6). To delete a channel, click **Delete** (Figure 42–6).
11. If needed, use the up and down arrows to prioritize channels. If available, the top-most channel receives messages meeting the filter criteria if you select **Send to the First Available Channel**.
12. Click **OK** to create the filter. Clicking **Cancel** discards the filter.

42.3.2 Editing a Filter

To edit a filter, first select it and then click **Edit** (Figure 42–9). The editing page appears for the filter, which enables you to add or change the filter properties described in [Section 42.3.1, "Creating Filters"](#).

42.3.3 Deleting a Filter

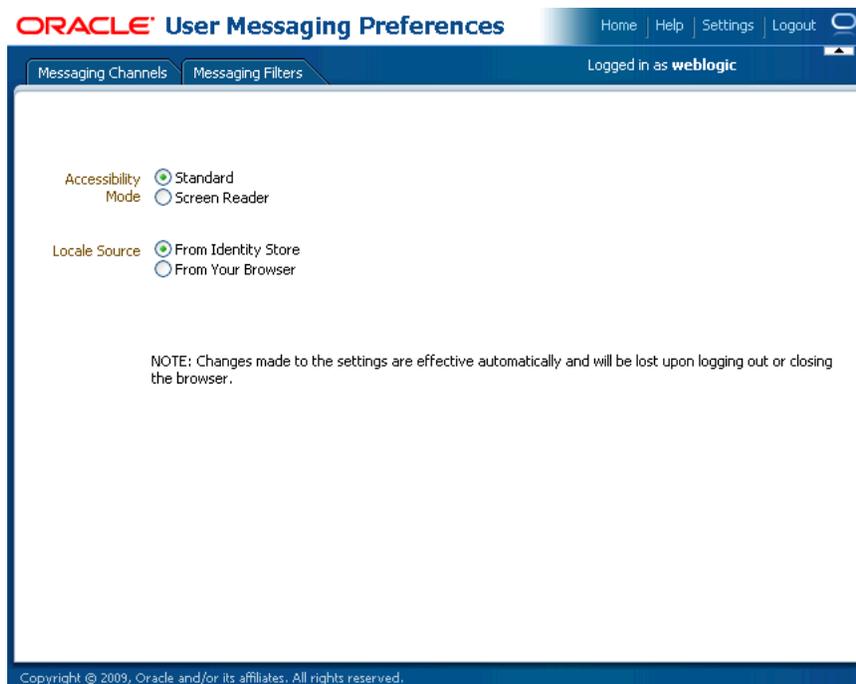
To delete a filter, first select it and then click **Delete** (Figure 42–6).

42.4 Configuring Settings

The **Settings** tab (Figure 42–10), accessed from the upper right area, enables users to set the following parameters:

- **Accessibility Mode:** select **Standard** or **Screen Reader**.
- **Locale Source:** select **From Identity Store** or **From Your Browser**.

Figure 42-10 Configuring Settings



Part VIII

Sharing Functionality Across Oracle SOA Suite Components

This part introduces functionality that is shared across components.

This part contains the following chapters:

- [Chapter 43, "Deploying SOA Composite Applications"](#)
- [Chapter 44, "Using Business Events and the Event Delivery Network"](#)
- [Chapter 45, "Creating Transformations with the XSLT Mapper"](#)
- [Chapter 46, "Working with Domain Value Maps"](#)
- [Chapter 47, "Working with Cross References"](#)
- [Chapter 48, "Using Two-Layer Business Process Management \(BPM\)"](#)
- [Chapter 49, "Testing SOA Composite Applications"](#)
- [Chapter 50, "Managing Policies"](#)
- [Chapter 51, "Defining Composite Sensors"](#)
- [Chapter 52, "Using Service Data Objects and Enterprise JavaBeans"](#)
- [Chapter 53, "Processing Large Documents"](#)

Deploying SOA Composite Applications

This chapter describes how to deploy SOA composite applications with Oracle JDeveloper and scripting tools and create configuration plans that enable you to move SOA composite applications to and from development, test, and production environments without having to redefine the URLs and properties of each environment.

This chapter includes the following sections:

- [Section 43.1, "Creating an Application Server Connection"](#)
- [Section 43.2, "Deploying a Single SOA Composite in Oracle JDeveloper"](#)
- [Section 43.3, "Deploying Multiple SOA Composite Applications in Oracle JDeveloper"](#)
- [Section 43.4, "Deploying and Using Shared Metadata Across SOA Composite Applications"](#)
- [Section 43.5, "Deploying an Existing SOA Archive in Oracle JDeveloper"](#)
- [Section 43.6, "Managing SOA Composite Applications with Scripts"](#)
- [Section 43.7, "Moving SOA Composite Applications to and from Development, Test, and Production Environments"](#)

See *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite* for instructions on deploying SOA composite applications from Oracle Enterprise Manager Fusion Middleware Control Console.

Note: Deployment from Oracle JDeveloper to a two-way, SSL-enabled Oracle WebLogic Server is not supported.

43.1 Creating an Application Server Connection

You must create a connection to the Oracle WebLogic Server to which to deploy a SOA composite application.

To create an application server connection:

1. From the **File** main menu, select **New**.
2. In the **General** list, select **Connections**.
3. Select **Application Server Connection**, and click **OK**.
4. In the **Connection Name** field, enter a name for the connection.
5. In the **Connection Type** list, select **WebLogic 10.3**.

6. Click **Next**.
7. In the **Username** field, enter the user authorized for access to the application server.
8. In the **Password** field, enter the password for this user.
9. Click **Next**.
10. In the **Weblogic Hostname** field, enter the host on which the Oracle WebLogic Server is installed.
11. In the **Port** and **SSL Port** fields, enter the appropriate port values.
12. If you want to use SSL, enable the **Always use SSL** checkbox.
13. In the **WLS Domain** field, enter the Oracle SOA Suite domain. For additional details about specifying domains, click **Help**.
14. Click **Next**.
15. Click **Test Connection** to test your server connection.
16. If the connection is successful, click **Finish**. Otherwise, click **Back** to make corrections in the previous dialogs.

43.2 Deploying a Single SOA Composite in Oracle JDeveloper

Oracle JDeveloper requires the use of profiles for SOA projects and applications to be deployed to Oracle WebLogic Server.

43.2.1 How to Deploy a Single SOA Composite

This section describes how to deploy a single SOA composite application with Oracle JDeveloper.

43.2.1.1 Optionally Creating a Project Deployment Profile

A required deployment profile is automatically created for your project. The application profile includes the JAR files of your SOA projects. If you want, you can create additional profiles.

To create a deployment profile:

1. In the Application Navigator, right-click the SOA project.
2. Select **Project Properties**.
The Project Properties dialog appears.
3. Click **Deployment**.
4. Click **New**.
The Create Deployment Profile dialog appears.
5. Enter the following values:

Table 43–1 Create Deployment Profile Dialog Fields and Values

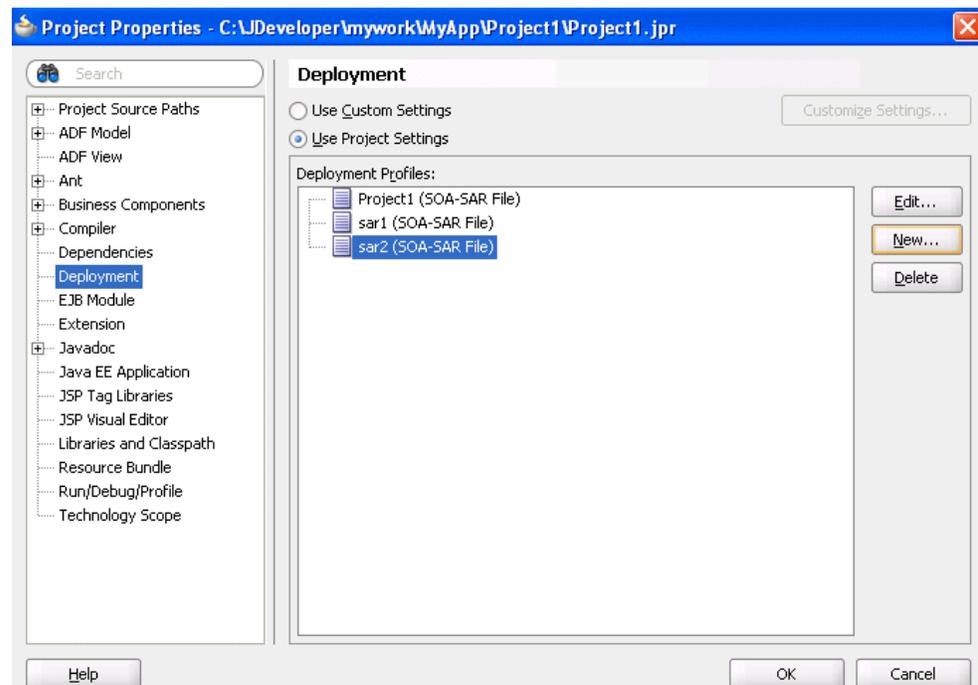
Field	Description
Archive Type	Select SOA-SAR File . A SOA archive (SAR) is a deployment unit that describes the SOA composite application. The SAR packages service components such as BPEL processes, business rules, human tasks, and mediator routing services into a single application. The SAR file is analogous to the BPEL suitcase archive of previous releases, but at the higher composite level and with any additional service components that your application includes (for example, human tasks, business rules, and mediator routing services).
Name	Enter a deployment profile name.

6. Click **OK**.

The SAR Deployment Profile dialog appears.

7. Click **OK** to close the SAR Deployment Profile Properties dialog.

The deployment profile shown in [Figure 43–1](#) displays in the Application Properties dialog.

Figure 43–1 Deployment Profile

43.2.1.2 Deploying the Profile

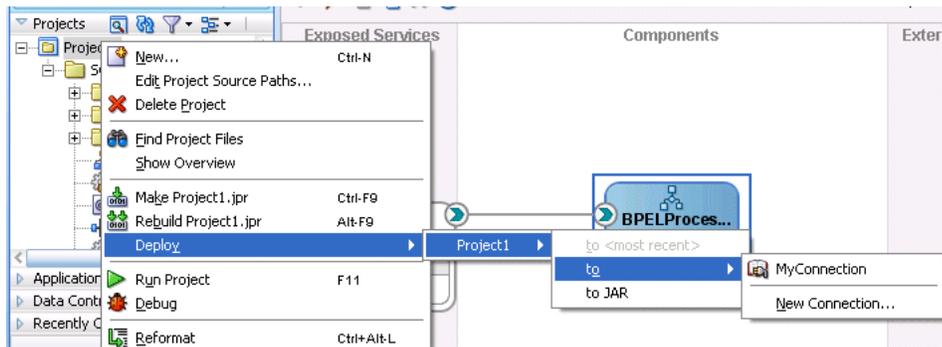
You now deploy the project profile to Oracle WebLogic Server. Deployment requires the creation of an application server connection. You can create a connection during deployment by selecting **New Connection** in Step 3 or before deployment by following the instructions in [Section 43.1, "Creating an Application Server Connection."](#)

To deploy the profile:

1. In the Application Navigator, right-click the SOA project.
2. Select **Deploy** > *deployment_profile_name* > **to**.

The value for *deployment_profile_name* is the SOA project name. [Figure 43–2](#) provides an example.

Figure 43–2 Project Profile Deployment



3. Select one of the following deployment options:

- **to JAR**

Creates a JAR file of the selected SOA project, but does *not* deploy it to Oracle WebLogic Server. This option is useful for environments in which:

- Oracle WebLogic Server may not be running, but you want to create the JAR file.
- You want to deploy multiple JAR files to Oracle WebLogic Server from a batch script. This option offers an alternative to opening all project profiles (which you may not have) and deploying them from Oracle JDeveloper.

- **to *Server_Connection_Name***

Creates a JAR file for the selected SOA project and deploys it to Oracle WebLogic Server. To deploy to Oracle WebLogic Server, you must first create a connection to it.

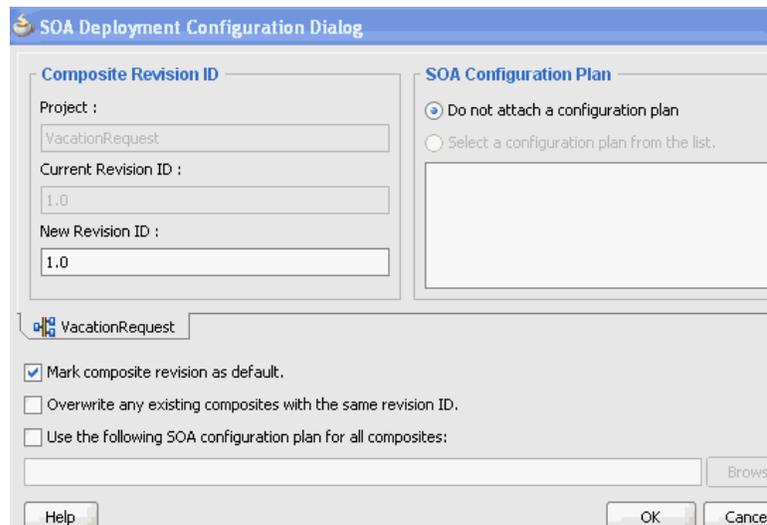
The SOA Deployment Configuration dialog that displays differs based on your selection.

4. Select the deployment option appropriate for your environment.

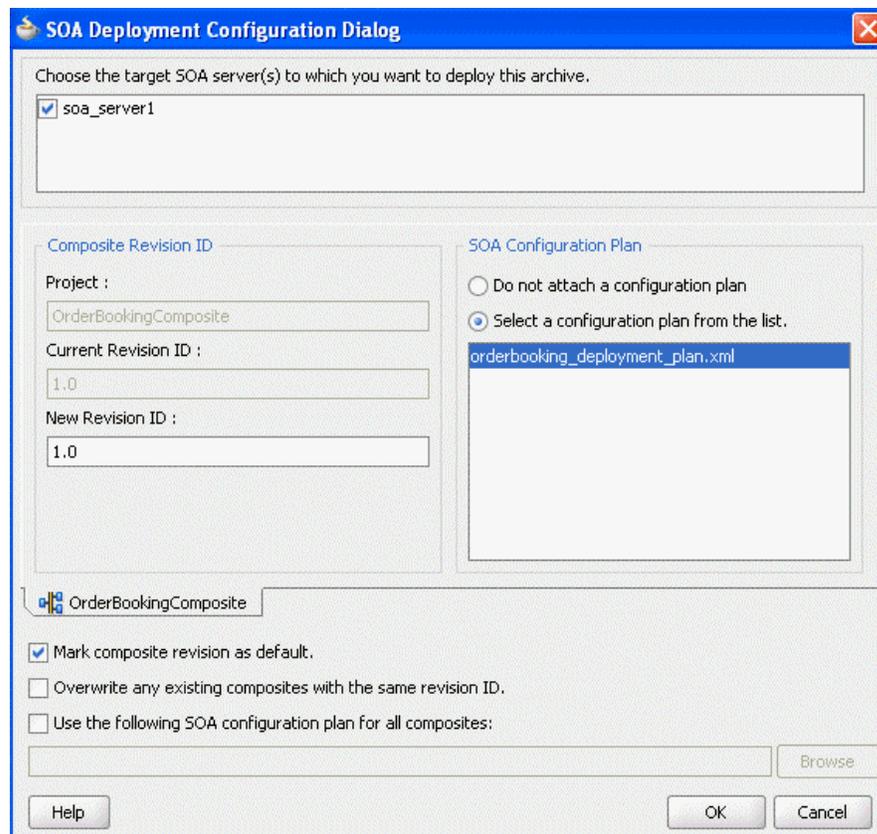
Table 43–2 Deployment Target

If You Select...	Go to...
to JAR	Step 4a
to <i>Server_Connection_Name</i>	Step 4b

- a. View the SOA Deployment Configuration dialog shown in [Figure 43–3](#).

Figure 43–3 SOA Deployment Configuration Dialog - JAR Deployment

- b. View the SOA Deployment Configuration dialog shown in [Figure 43–4](#).

Figure 43–4 SOA Deployment Configuration Dialog - Server Deployment

5. Provide values appropriate to your environment. If you selected to deploy to a server, an additional field for selecting the server to which to deploy is displayed at the top of the dialog.

Table 43–3 SOA Deployment Configuration Dialog

Field	Description
Choose the target SOA server(s) to which you want to deploy this archive	Note: This option only displays if you selected to deploy to a server. If there are multiple servers or cluster nodes, select to deploy to one or more servers or nodes.
Project	Displays the project name.
Current Revision ID	Displays the current revision ID of the project.
New Revision ID	Optionally change the revision ID of the SOA composite application.
Mark composite revision as default	If you do not want the new revision to be the default, you can uncheck this box. By default, a newly deployed composite revision is the default. This revision is instantiated when a new request comes in.
Overwrite any existing composites with the same revision ID	Select to overwrite any existing SOA composite application of the same revision value.
Use the following SOA configuration plan for all composites	Click Browse to select the same configuration plan to use for all composite applications. This option is used when deploying multiple composite applications.
Do not attach configuration plan	Select to not include a configuration plan with the SOA composite application JAR file. If you have not created a configuration plan, this field is disabled.
Select a configuration plan from the list	Select to include a configuration plan with the SOA composite application. The configuration plan enables you to define the URL and property values to use in different environments. During process deployment, the configuration plan is used to search the SOA project for values that must be replaced to adapt the project to the next target environment. If you have not created a configuration plan, this field is disabled. See Section 43.7, "Moving SOA Composite Applications to and from Development, Test, and Production Environments" for instructions on creating a configuration plan.

6. Click **OK**.
7. View the messages that display in the Deployment log window at the bottom of Oracle JDeveloper.

If deployment is successful, a JAR file for the SOA project is created under the **deploy** folder in Oracle JDeveloper with a naming convention of **sca_composite_name_revrevision_number.jar**.

You are now ready to monitor your application from Oracle Enterprise Manager Grid Control Console. See *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite* for details.

If deployment is unsuccessful, view the messages that display in the Deployment log window and take corrective actions.

Note: If you want to redeploy the same version of a SOA composite application, you cannot change the composite name. You can deploy with the same revision number if you selected the **Overwrite any existing composites with the same revision ID** checkbox on the SOA Deployment Configuration dialog.

43.2.2 What You May Need to Know About Oracle JDeveloper Deployment to a Managed Oracle WebLogic Server

If you start a managed Oracle WebLogic Server without starting an Oracle WebLogic Administration Server (known as running in independence mode) and attempt to deploy a SOA composite application from Oracle JDeveloper, you receive the following error:

```
Deployment cannot continue! No SOA Configured target servers found
```

The Oracle WebLogic Administration Server must be running. Deployment uses the Oracle WebLogic Administration Server connection to identify the servers running Oracle SOA Suite. In addition, do not create an application server connection to a managed Oracle WebLogic Server; only create connections to an Oracle WebLogic Administration Server.

You can also receive a similar error if the condition of the SOA-configured Oracle WebLogic Server is not healthy. This condition displays in the **Health** column of the Servers page of Oracle WebLogic Server Administration Console.

Note that you can use the WebLogic Scripting Tool (WLST) to deploy SOA composite applications to a managed Oracle WebLogic Server without starting an Oracle WebLogic Administration Server. See [Section 43.6.1, "How to Manage SOA Composite Applications with the WLST Utility"](#) for details.

43.2.3 What You May Need to Know About Invoking References in One-Way SSL Environments in Oracle JDeveloper

When invoking a web service as an external reference from a SOA composite application in one-way SSL environments, ensure that the certificate name (CN) and the hostname of the server exactly match. This ensures a correct SSL handshake.

For example, if a web service is named `adfbc` and the certificate has a server name of `myhost05`, the following results in an SSL handshake exception.

```
<import namespace="/adfbc1/common/"

@ location="https://myhost05.us.oracle.com:8002/CustomApps-adfbc1-context-root/AppModuleService?WSDL"
  importType="wsdl"/>
<import namespace="/adfbc1/common/" location="Service1.wsdl"
  importType="wsdl"/>
```

If you switch the order of `import`, the SSL handshake passes.

```
<import namespace="/adfbc1/common/" location="Service1.wsdl"
  importType="wsdl"/>
<import namespace="/adfbc1/common/"

@ location="https://myhost05.us.oracle.com:8002/CustomApps-adfbc1-context-root/AppModuleService?WSDL"
  importType="wsdl"/>
```

Note the following restrictions around this issue:

- There are no options for ignoring hostname verification in Oracle JDeveloper as exist with the Oracle WebLogic Server Administration Console. This is because the SSL kit used by Oracle JDeveloper is different. Only the trust store can be configured from the command line. All other certificate arguments are not passed.
- In the WSDL file, `https://hostname` must match with that in the certificate, as described above. You cannot perform the same procedures as you can with a browser. For example, if the hostname is `myhost05.us.oracle.com` in the certificate's CN, then you can use `myhost05`, `myhost05.us.oracle.com`, or the IP address from a browser. In Oracle JDeveloper, always use the same name as in the certificate (that is, `myhost05.us.oracle.com`).

43.3 Deploying Multiple SOA Composite Applications in Oracle JDeveloper

You can deploy multiple SOA composite applications to Oracle WebLogic Server at the same time by using the SOA bundle profile. This profile enables you to include one or more SAR profiles in the bundle and deploy the bundle to Oracle WebLogic Server.

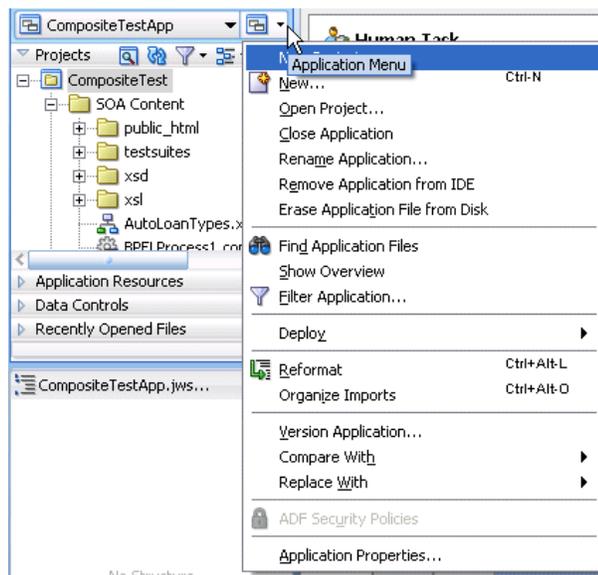
Note: You cannot deploy multiple SOA applications that are dependent upon one another in the same SOA bundle profile. For example, if application A calls application B, then you must first deploy application B separately.

43.3.1 How to Deploy Multiple SOA Composite Applications

To deploy multiple SOA composite applications

1. From the **Application** menu, select **Application Properties**, as shown in [Figure 43–5](#).

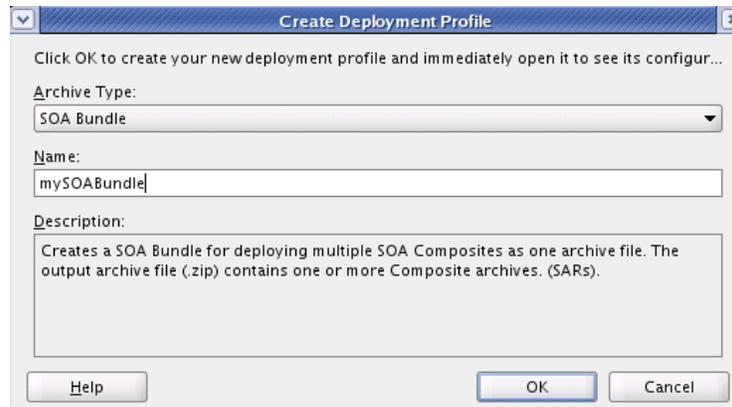
Figure 43–5 Application Properties



2. In the **Application Properties** dialog, click **Deployment**.
3. Click **New**.
The Create Deployment Profile dialog appears.
4. In the **Archive Type** list, select **SOA Bundle**.
5. In the **Name** field, enter a name.

Figure 43–6 provides details.

Figure 43–6 Select the SOA Bundle



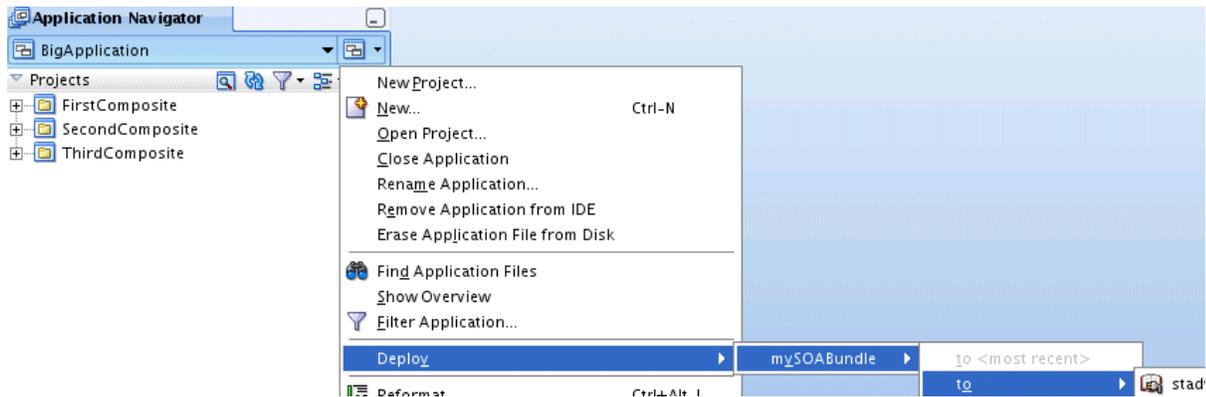
6. Click **OK**.
7. In the navigator on the left, select the **Dependencies** node.
8. Select the SARs you want to include in this bundle, as shown in Figure 43–7.

Figure 43–7 Select the SAR



9. Click **OK**.
10. Click **OK** to close the Application Properties dialog.
11. Select the **Application** menu again, then select **Deploy > SOA_bundle_name > to**. [Figure 43–8](#) provides details.

Figure 43–8 Deployment of Application



12. Select one of the following options:
 - **to *Server_Connection_Name***
Creates a ZIP file of the application deployment profile that includes JAR files of all selected SOA projects and deploys it to Oracle WebLogic Server. To deploy to Oracle WebLogic Server, you must first create a connection to it.
 - **to ZIP**
Creates a ZIP file of the application deployment profile that includes JAR files of all selected SOA projects, but does *not* deploy it to Oracle WebLogic Server. This option is useful for environments in which:
 - Oracle WebLogic Server may not be running.
 - You want to deploy multiple JAR files to Oracle WebLogic Server from a batch script. This option offers an alternative to opening all application profiles (which you may not have) and deploying them from Oracle JDeveloper.

The SOA Deployment Configuration dialog that displays is based on your selection:

 - If you selected to deploy to a ZIP file, the SOA Deployment Configuration dialog shown in [Figure 43–3](#) appears.
 - If you selected to deploy to the server, the SOA Deployment Configuration dialog shown in [Figure 43–4](#) appears.
13. See Step 5 on page 43-6 for details about responses to provide.
14. Click **OK**.

43.4 Deploying and Using Shared Metadata Across SOA Composite Applications

This section describes how to deploy and use shared metadata across SOA composite applications.

43.4.1 How to Deploy Shared Metadata

Shared metadata is deployed to the SOA Infrastructure on the application server as a JAR file. The JAR file should contain all the resources to share. In Oracle JDeveloper, you can create a JAR profile for creating a shared artifacts archive.

All shared metadata is deployed to an existing SOA Infrastructure partition on the server. This metadata is deployed under the `/apps` namespace. For example, if you have a `MyProject/xsd/MySchema.xsd` file in the JAR file, then this file is deployed under the `/apps` namespace on the server. When you refer to this artifact in Oracle JDeveloper using an SOA-MDS connection, the URL becomes `oramds:/apps/MyProject/xsd/MySchema.xsd`.

This section describes how to perform the following tasks:

- Create a JAR profile and include the artifacts to share
- Create a SOA bundle that includes the JAR profile
- Deploy the SOA bundle to the application server

43.4.1.1 Create a JAR Profile and Include the Artifacts to Share

To create a JAR profile and include the artifacts to share:

1. In the Application Navigator, right-click the SOA project.

2. Select **Project Properties**.

The Project Properties dialog appears.

3. Click **Deployment** in the navigational tree on the left.

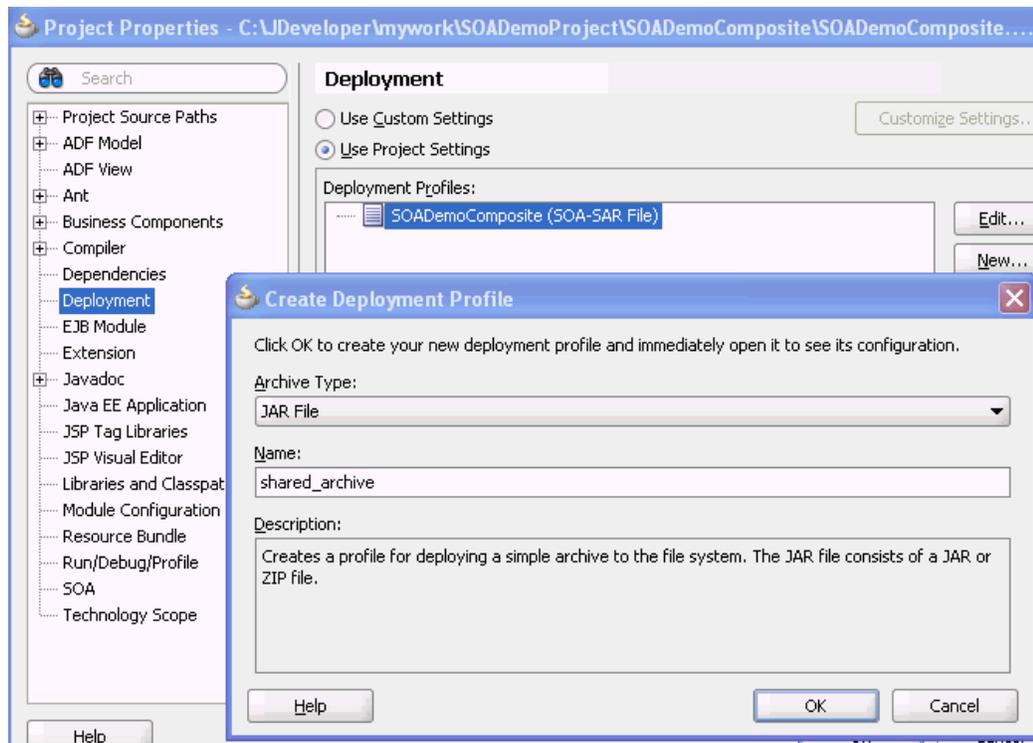
4. Click **New**.

The Create Deployment Profile dialog appears.

5. From the **Archive Type** list, select **JAR File**.

6. In the **Name** field, enter a name (for this example, `shared_archive` is entered).

The Create Deployment Profile looks as shown in [Figure 43-9](#).

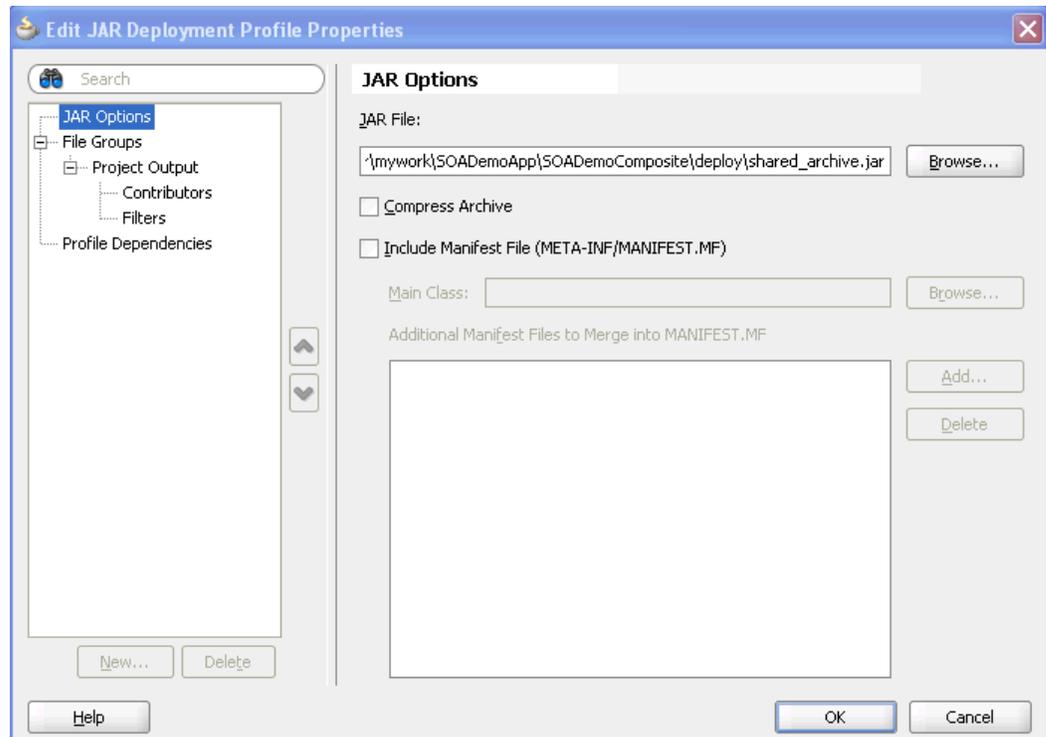
Figure 43–9 JAR File Selection

7. Click **OK**.

The JAR Deployment Profile Properties dialog appears.

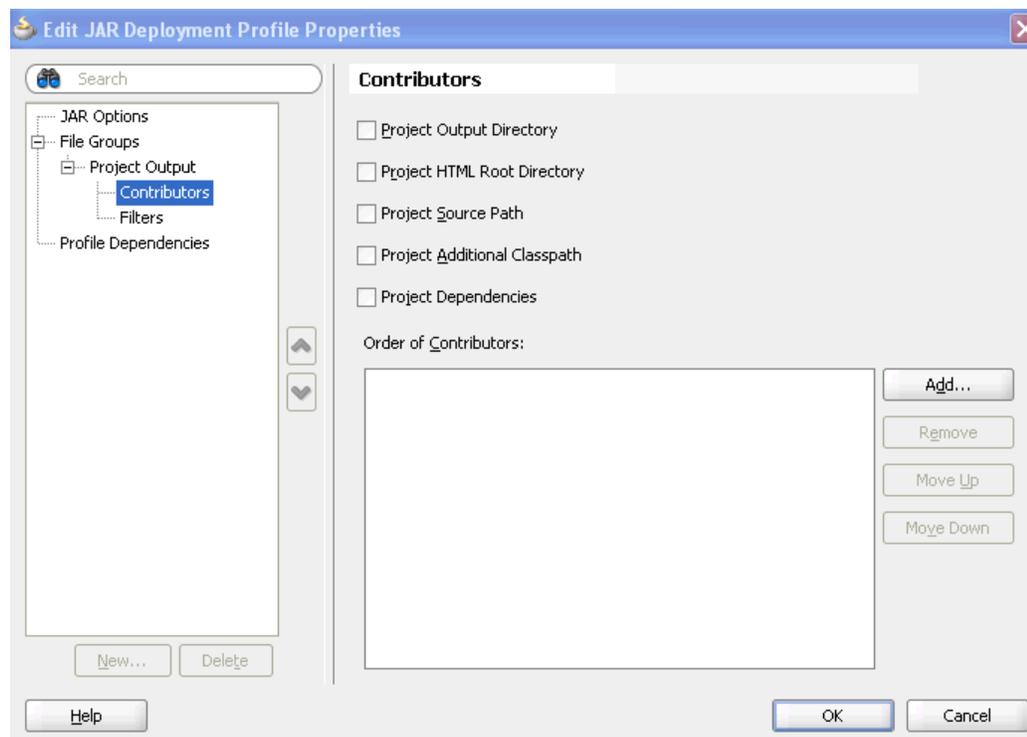
8. Select **JAR Options** from the navigational tree on the left.
9. Deselect **Include Manifest File (META-INF/MANIFEST.MF)**, as shown in [Figure 43–10](#).

This prevents the archive generator from adding the manifest file (META-INF/MANIFEST.MF) into the JAR file.

Figure 43–10 JAR File Options

10. Select **File Groups > Project Output > Contributors** from the navigational tree on the left.
11. Deselect the **Project Output Directory** and **Project Dependencies** options, as shown in [Figure 43–11](#).

This prevents the archive generator from adding the contents of the project output and project dependencies into the archive.

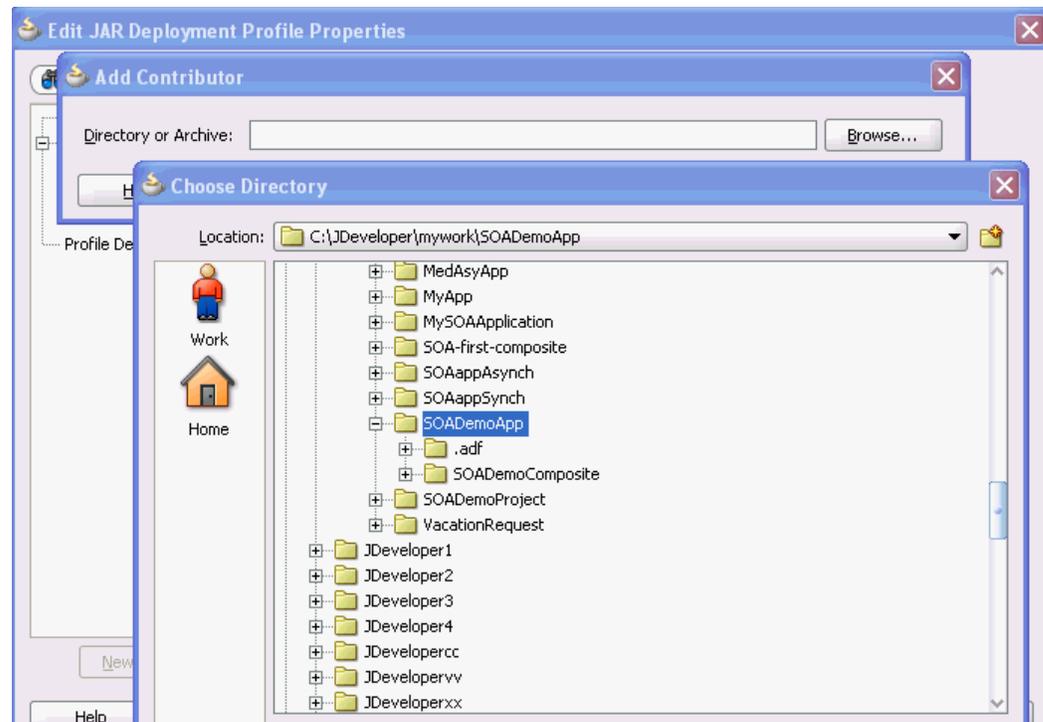
Figure 43–11 Contributors

12. Click **Add** to add a new contributor.

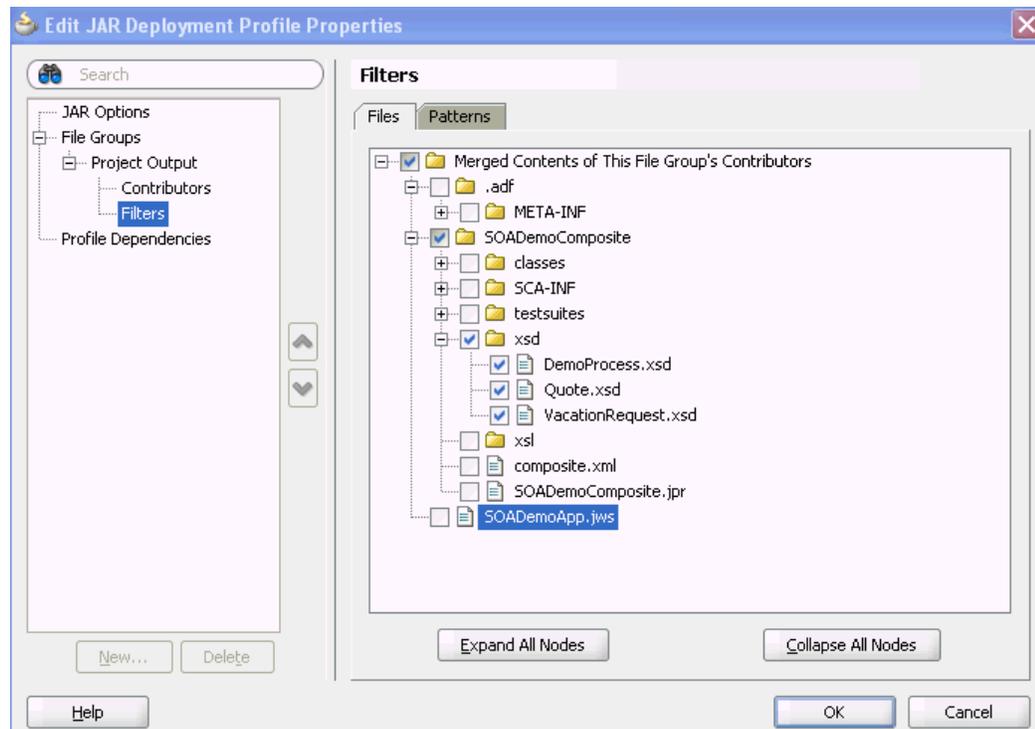
The Add Contributor dialog appears. This dialog enables you to add artifacts to your archive.

13. Click **Browse**.

14. Select the folder in which your artifacts reside, as shown in [Figure 43–12](#). Note that this also determines the hierarchy of artifacts in the archive.

Figure 43–12 Artifact Selection

15. Click **Select** to close the Choose Directory dialog.
16. Click **OK** to close the Add Contributor dialog.
17. Select **File Groups > Project Output > Filters** from the navigational tree on the left.
18. Select only the artifacts to include in the archive, as shown in [Figure 43–13](#). For this example, the archive contains the following XSD files:
 - **SOADemoComposite/xsd/DemoProcess.xsd**
 - **SOADemoComposite/xsd/Quote.xsd**
 - **SOADemoComposite/xsd/VacationRequest.xsd**

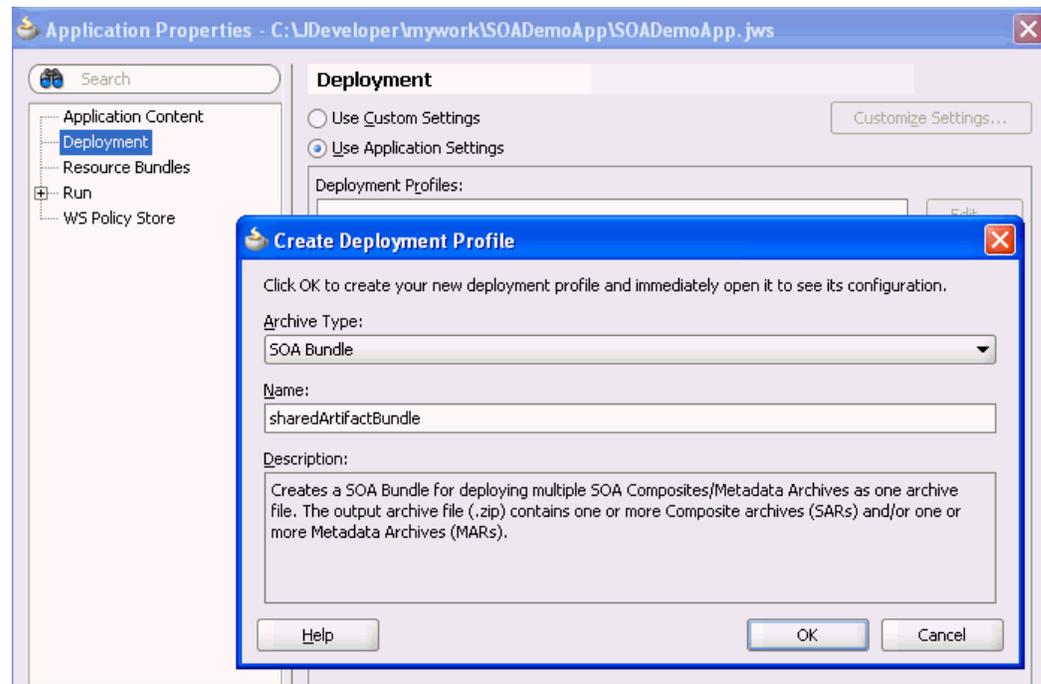
Figure 43–13 Artifacts to Include in the Archive

19. Click **OK** to save changes to the JAR deployment profile.
20. Click **OK** to save the new deployment profile.
21. From the **File** main menu, select **Save All**.

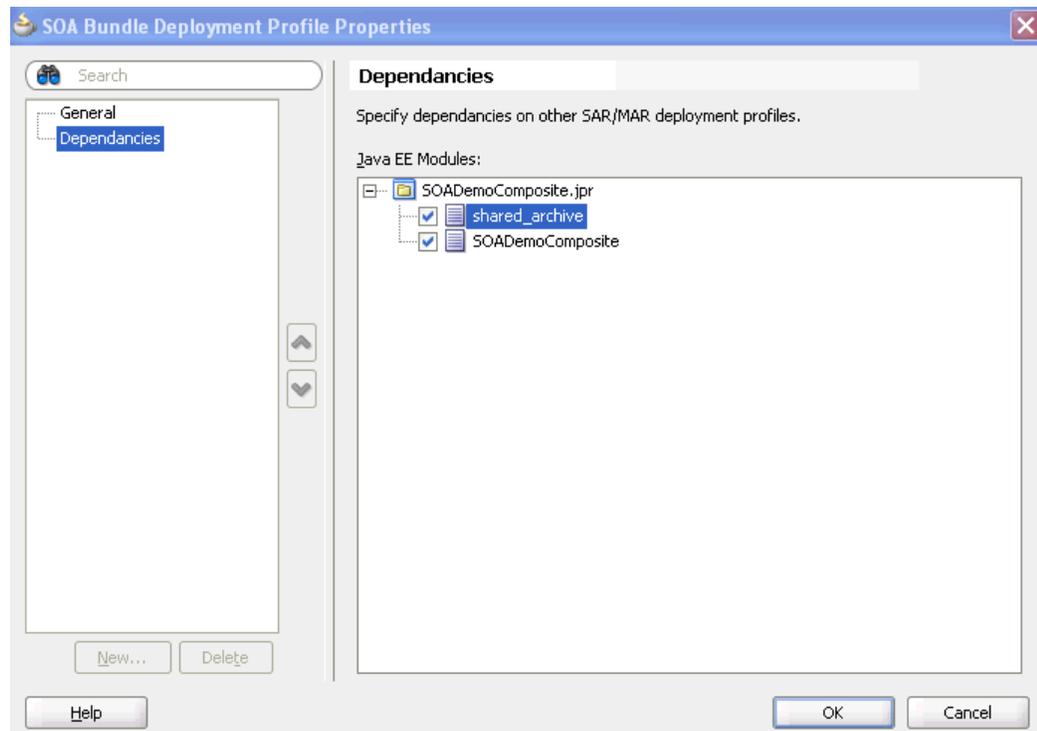
43.4.1.2 Create a SOA Bundle that Includes the JAR Profile

To create a SOA bundle that includes the JAR profile:

1. From the Application Menu, select **Application Properties > Deployment**.
2. Click **New** to create a SOA bundle profile.
The Create Deployment Profile dialog appears.
3. From the **Archive Type** list, select **SOA Bundle**. A bundle is a collection of multiple SOA composite applications.
4. In the **Name** field, enter a name (for this example, `sharedArtifactBundle` is entered).

Figure 43–14 SOA Bundle Creation

5. Click **OK**.
6. Select **Dependencies** from the navigational tree on the left.
7. Select the JAR file and SOA-SAR profiles you previously created (for this example, named **shared_archive** and **sharedArtifactBundle**, respectively). You have the option of a JAR, an SOA-SAR, or both.

Figure 43–15 Deployment Profile Dependencies

8. Click **OK** to save the SOA bundle deployment profile changes.
9. Click **OK** to save the new deployment profile.
10. From the **File** main menu, select **Save All**.

43.4.1.3 Deploy the SOA Bundle

To deploy the SOA bundle:

1. Right-click the **Application** menu and select **Deploy > SOA_bundle_name > to server_connection**. If you do not have an existing connection, you must first create one.

This deploys the SOA bundle to the application server (shared artifacts are deployed to the MDS database of Oracle SOA Suite).

43.4.2 How to Use Shared Metadata

This section describes how to browse and select the shared metadata you created in [Section 43.4.1, "How to Deploy Shared Metadata."](#)

43.4.2.1 Create a SOA-MDS Connection

To create a SOA-MDS connection:

1. From the **File** menu, select **New > Connections > SOA-MDS Connection**.
2. In the Welcome page, click **Next**.
3. In the **Connection Name** field, enter a name.
4. From the **Connection Type** list, select **DB based MDS**.

5. Click **Next**.

The Connection Type page appears.

6. Select an existing connection or create a new connection to the Oracle SOA Suite database with the MDS schema.

7. From the **Select MDS partition** list, select the MDS partition (for example, **soa-infra**).

8. Click **Next**.

9. Click **Finish**.

You can now browse the connection in the Resource Palette and view shared artifacts under the **/apps** node.

43.4.2.2 Create a BPEL Process

You can now browse and use the shared metadata from a different SOA composite application. For this example, you create a BPEL process service component in a different application.

To create a BPEL process:

1. Create a new BPEL process service component in a different application. For more information, see [Section 5.1.1, "How to Add a BPEL Process Service Component."](#)

2. In the Create BPEL Process dialog, click the **Browse** icon to the right of the **Input** field.

The Type Chooser dialog appears.

3. In the upper right corner, click the **Import Schema File** icon.

The Import Schema File dialog appears.

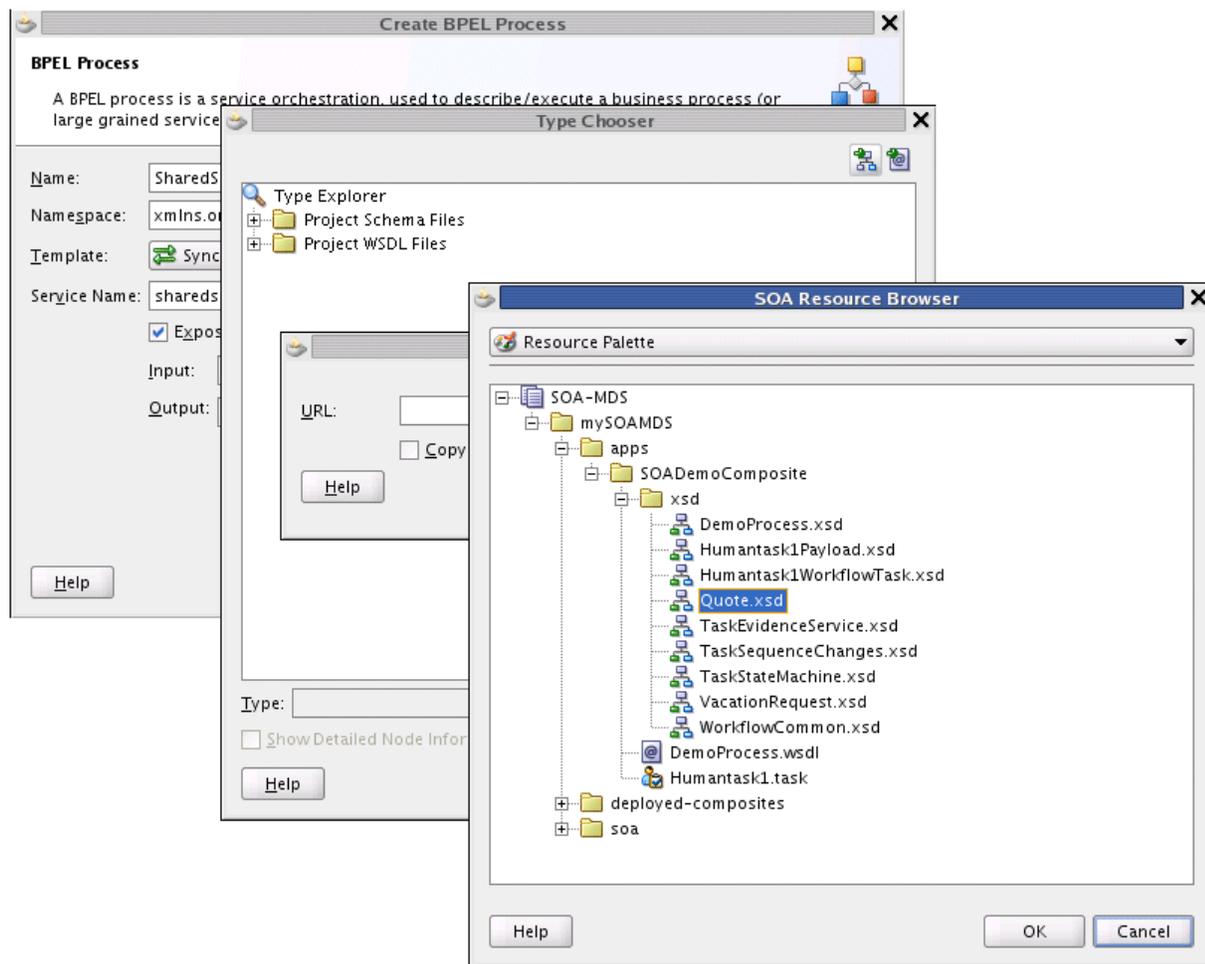
4. To the right of the **URL** field, click the **Browse** icon.

The SOA Resource Browser dialog appears.

5. At the top of the dialog, select **Resource Palette** from the list.

6. Select shared metadata, as shown in [Figure 43-16](#). For this example, the **Quote.xsd** file that you selected to include in the archive in Step 18 of [Section 43.4.1.1, "Create a JAR Profile and Include the Artifacts to Share"](#) is selected.

Figure 43–16 Shared Metadata in the SOA Resource Browser



7. Click **OK**.
8. In the Import Schema File dialog, click **OK**.
9. In the Type Chooser dialog, select a node of **Quote.xsd** (for this example, **QuoteRequest**), and click **OK**.
10. In the Create BPEL Process dialog, click **OK** to complete creation.
11. In the Application Navigator, select the WSDL file for the BPEL process.
12. Click **Source**.

The WSDL file includes the following definition.

```
<wsdl:types>
  <schema xmlns="http://www.w3.org/2001/XMLSchema">
    <import namespace="http://www.mycompany.com/ns/salesquote"
      schemaLocation="orams:/apps/SOADemoComposite/xsd/Quote.xsd" />
  </schema>
</wsdl:types>
```

13. Continue modeling the BPEL process as necessary.
14. Deploy the SOA composite application that includes the BPEL process.

43.5 Deploying an Existing SOA Archive in Oracle JDeveloper

You can deploy an existing SOA archive from the Application Server Navigator in Oracle JDeveloper.

Notes:

- The archive must exist. You cannot create an archive in the Deploy SOA Archive dialog.
 - These instructions assume you have created an application server connection to an Oracle WebLogic Administration Server on which the SOA Infrastructure is deployed. Creating a connection to an Oracle WebLogic Administration Server enables you to browse for SOA composite applications deployed in the same domain. From the **File** main menu, select **New > Connections > Application Server Connection** to create a connection.
-
-

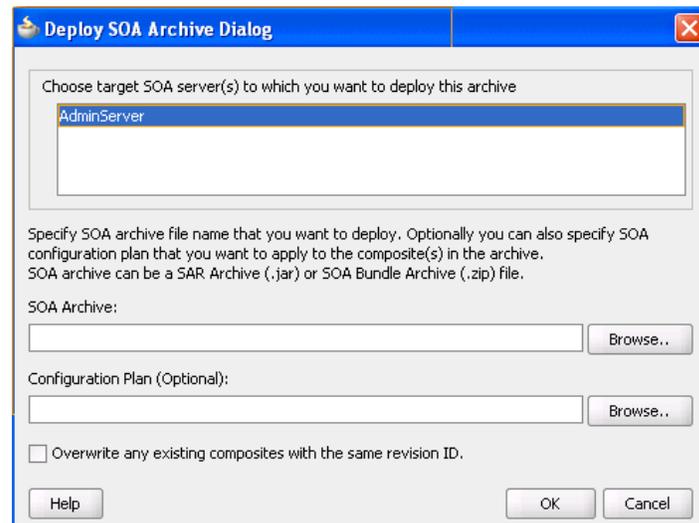
43.5.1 How to Deploy an Existing SOA Archive from Oracle JDeveloper

To Deploy an Existing SOA Archive from Oracle JDeveloper:

1. From the **View** menu, select **Application Server Navigator**.
2. Expand your connection name (for this example, named **myConnection**).
3. Right-click the **SOA** folder.
4. Select **Deploy SOA Archive**.

The Deploy SOA Archive dialog shown in [Figure 43–17](#) appears.

Figure 43–17 Deploy SOA Archive



5. Provide responses appropriate to your environment, as shown in [Table 43–4](#).

Table 43–4 Create Deployment Profile Dialog Fields and Values

Field	Description
Choose target SOA server(s) to which you want to deploy this archive	Select the Oracle WebLogic Administration Server to which to deploy the archive.
SOA Archive	Click Browse to select a <i>prebuilt</i> SOA composite application archive. The archive consists of a JAR file of a single application or a SOA bundle ZIP file containing multiple applications.
Configuration Plan (Optional)	Click Browse to select a configuration plan to attach to the SOA composite application archive. The configuration plan enables you to define the URL and property values to use in different environments. During process deployment, the configuration plan is used to search the SOA project for values that must be replaced to adapt the project to the next target environment. For information about creating configuration plans, see Section 43.7.4, "How to Create a Configuration Plan in Oracle JDeveloper" or Section 43.7.5, "How to Create a Configuration Plan with the WLST Utility."
Overwrite any existing composites with the same revision ID	Select to overwrite (redeploy) an existing SOA composite application with the same revision ID. The consequences of this action are as follows: <ul style="list-style-type: none"> ■ A new version 1.0 of the SOA composite application is redeployed, overwriting a previously deployed 1.0 version. ■ The older, currently-deployed version of this revision is removed (overwritten). ■ If the older, currently-deployed version of this revision has running instances, the state of those instances is changed to stale.

6. Click OK.

43.6 Managing SOA Composite Applications with Scripts

You can also manage SOA composite applications from a command line or scripting environment using WLST or `ant`. These options are well-suited for automation and can be easily integrated into existing release processes.

43.6.1 How to Manage SOA Composite Applications with the WLST Utility

You can manage SOA composite applications with the WLST scripting utility. For instructions, see *Oracle Fusion Middleware WebLogic Scripting Tool Command Reference*.

43.6.2 How to Manage SOA Composite Applications with ant Scripts

You can manage SOA composite applications with the `ant` utility. `ant` is a Java-based build tool used by Oracle SOA Suite for managing SOA composite applications. The configuration files are XML-based and call out a target tree where various tasks are executed.

[Table 43–5](#) lists the `ant` scripts available in the `Middleware_Home\SOA_Suite_Home\bin` directory.

Table 43–5 *ant Management Scripts*

Script	Description
<code>ant-sca-test.xml</code>	Attaches, extracts, generates, and validates configuration plans for a SOA composite application.
<code>ant-sca-compile.xml</code>	Compiles a SOA composite application.
<code>ant-sca-package.xml</code>	Packages a SOA composite application into a composite SAR file.
<code>ant-sca-deploy.xml</code>	Deploys a SOA composite application.
<code>ant-sca-deploy.xml</code> <code>undeploy</code>	Undeploys a SOA composite application.
<code>ant-sca-mgmt.xml</code>	Manages a SOA composite application, including starting, stopping, activating, retiring, assigning a default revision version, and listing deployed SOA composite applications.
<code>ant-sca-upgrade.xml</code>	Migrates BPEL and ESB release 10.1.3 metadata to release 11g. Note: If any Java code is part of the project, you must manually modify the code to pass compilation with an 11g compiler. For BPEL process instance data, active data used by the 10.1.3 Oracle BPEL Server is not migrated.

For additional information about `ant`, visit the following URL:

<http://ant.apache.org>

43.6.2.1 Testing a SOA Composite Application

Example 43–1 provides an example of executing a test case. Test cases enable you to automate the testing of SOA composite applications.

Example 43–1 *Testing an Application*

```
ant -f ant-sca-test.xml -Dscatest.input=MyComposite
-Djndi.properties=/home/jdoe/jndi.properties
```

Argument	Definition
<code>scatest</code>	<p>Possible inputs are as follows:</p> <ul style="list-style-type: none"> ■ <code>java.passed.home</code> The script picks this from the environment value of <code>JAVA_HOME</code>. Provide this input to override. ■ <code>wl_home</code> This is the location of Oracle WebLogic Server home (defaults to <code>Oracle_Home/.../wlserver_10.3</code>). ■ <code>scatest.input</code> The name of the composite to test. ■ <code>scatest.format</code> The format of the output file (defaults to <code>native</code>; the other option is <code>junit</code>). ■ <code>scatest.result</code> The result directory in which to place the output files (defaults to <code>temp_dir/out</code>). ■ <code>jndi.properties.input</code> The <code>jndi.properties</code> file to use.

Argument	Definition
jndi.properties	<p>Absolute path to the JNDI property file. This is a property file that contains JNDI properties for connecting to the server. For example:</p> <pre>java.naming.factory.initial=weblogic.jndi.WLInitialContextFactory java.naming.provider.url=t3://myserver.us.oracle.com:8001/soa-infra java.naming.security.principal=weblogic dedicated.connection=true dedicated.rmicontext=true</pre> <p>Since a composite test (in a test suite) is executed on the SOA Infrastructure, this properties file contains the connection information. For this example, these properties create a connection to the SOA Infrastructure hosted in <code>myserver.us.oracle.com</code>, port 8001 and use a user name of <code>weblogic</code>. You are prompted to specify the password.</p> <p>You typically create one <code>jndi.properties</code> file (for example, in <code>/home/myhome/jndi.properties</code>) and use it for all test runs.</p>

For more information on creating and running tests on SOA composite applications, see [Chapter 49, "Testing SOA Composite Applications"](#) and *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite*.

43.6.2.2 Compiling a SOA Composite Application

[Example 43–2](#) provides an example of compiling a SOA composite application, which validates it for structure and syntax.

Example 43–2 Compiling an Application

```
ant -f ant-sca-compile.xml
-Dscac.input=/myApplication/myComposite/composite.xml
```

Argument	Definition
scac	<p>Possible inputs are as follows:</p> <ul style="list-style-type: none"> ■ <code>java.passed.home</code> The script picks this from the environment value of <code>JAVA_HOME</code>. Provide this input to override. ■ <code>wl_home</code> This is the location of Oracle WebLogic Server home (defaults to <code>Oracle_Home/.../wlserver_10.3</code>). ■ <code>scac.input</code> The <code>composite.xml</code> file to compile. ■ <code>scac.output</code> The output file with <code>scac</code> results (defaults to <code>temp_dir/out.xml</code>). ■ <code>scac.error</code> The file with <code>scac</code> errors (defaults to <code>temp_dir/out.err</code>). ■ <code>scac.application.home</code> The application home directory of the composite being compiled. ■ <code>scac.displayLevel</code> Controls the level of logs written to <code>scac.output</code> file. The value can be 1, 2, or 3 (this defaults to 1).

43.6.2.3 Packaging a SOA Composite Application into a Composite SAR file

[Example 43–3](#) provides an example of packaging a SOA composite application into a composite SAR file. The outcome of this command is a SOA archive. Check the output of the command for the exact location of the resulting file.

Example 43–3 Packaging an Application

```
ant -f ant-sca-package.xml
-DcompositeDir=C:\demo\end2end-105-POProcessing\po\solutions\ch9\POProcessing\POPr
ocessing
-DcompositeName=POProcessing
-Drevision=6-cmdline
-Dsca.application.home=C:\demo\end2end-105-POProcessing\po\solutions\ch9\POProces
sing
```

Argument	Definition
compositeDir	Absolute path of a directory that contains composite artifacts.
compositeName	Name of the composite.
revision	Revision ID of the composite.
sca.application.home	Optional. Absolute path of the application home directory. This property is required if you have shared data.
oracle.home	Optional. The oracle.home property.

43.6.2.4 Deploying SOA Composite Application

[Example 43–4](#) provides an example of deploying a SOA composite application.

Example 43–4 Deploying an Application

```
ant -f ant-sca-deploy.xml
-DserverURL=http://localhost:8001
-DsarLocation=C:\demo\end2end-105-POProcessing\po\solutions\ch9\POProcessing\POPro
cessing\deploy\sca_POProcessing_rev6-cmdline.jar
-Doverwrite=true
-Duser=weblogic
-DforceDefault=true
-Dconfigplan=C:\demo\end2end-105-POProcessing\po\solutions\ch9\POProcessing\POProc
essing\demed_cfgplan.xml
```

Note: After specifying the user name, enter the password when prompted.

Argument	Definition
serverURL	URL of the server that hosts the SOA Infrastructure application (for example, <code>http://myhost10:8001</code>).
sarLocation	Absolute path to one the following: <ul style="list-style-type: none"> ▪ SAR file. ▪ ZIP file that includes multiple SARs.

Argument	Definition
overwrite	Optional. Indicates whether to overwrite an existing SOA composite application on the server. <ul style="list-style-type: none"> ▪ false (default): Does not overwrite the file. ▪ true: Overwrites the file.
user	Optional. User name to access the composite deployer servlet when basic authentication is configured.
password	Optional. Password to access the composite deployer servlet when basic authentication is configured. If you enter the user name, you are prompted to enter the password if you do not provide it here.
forceDefault	Optional. Indicates whether to set the version being deployed as the default version for that composite application. <ul style="list-style-type: none"> ▪ true (default): Makes it the default composite. ▪ false: Does not make it the default composite.
configplan	Absolute path of a configuration plan to be applied to a specified SAR file or to all SAR files included in the ZIP file.
sysPropFile	Passes in a system properties file that is useful for setting extra system properties, for debugging, for SSL configuration, and so on. If you specify a file name (for example, <code>tmp-sys.properties</code>), you can define properties such as the following: <code>javax.net.debug=all</code>

43.6.2.5 Undeploying a SOA Composite Application

[Example 43–5](#) provides an example of undeploying a SOA composite application.

Example 43–5 Undeploying a SOA Composite Application

```
ant -f ant-sca-deploy.xml undeploy
-DserverURL=http://localhost:8001
-DcompositeName=POProcessing
-Drevision=rev6-cmdline
-Duser=weblogic
```

Note: After specifying the user name, enter the password when prompted.

Argument	Definition
serverURL	URL of the server that hosts the SOA Infrastructure application (for example, <code>http://myhost10:7001</code>).
compositeName	Name of the SOA composite application.
revision	Revision ID of the SOA composite application.
user	Optional. User name to access the composite deployer servlet when basic authentication is configured. If you enter the user name, you are prompted to enter the corresponding password.
password	Optional. Password to access the composite deployer servlet when basic authentication is configured.

43.6.2.6 Managing a SOA Composite Application

Example 43–6 through Example 43–11 provide examples of managing a SOA composite application.

Example 43–6 Starting an Application

```
ant -f ant-sca-mgmt.xml startComposite -Dhost=myhost -Dport=8001 -Duser=weblogic
-DcompositeName=HelloWorld -Drevision=1.0
```

Note: After specifying the user name, enter the password when prompted.

Argument	Definition
host	Hostname of the Oracle WebLogic Server (for example, myhost).
port	Port of the Oracle WebLogic Server (for example, 7001).
user	User name for connecting to the running server to get MBean information (for example, weblogic).
password	Password for the user name.
compositeName	Name of the SOA composite application.
revision	Revision of the SOA composite application.
label	Optional. Label of the SOA composite application. The label identifies the metadata service (MDS) artifacts associated with the application. If the label is not specified, the system finds the latest one.

Example 43–7 Stopping an Application

```
ant -f ant-sca-mgmt.xml stopComposite -Dhost=myhost -Dport=8001 -Duser=weblogic
-DcompositeName=HelloWorld -Drevision=1.0
```

Note: After specifying the user name, enter the password when prompted.

Argument	Definition
host	Hostname of the Oracle WebLogic Server (for example, myhost).
port	Port of the Oracle WebLogic Server (for example, 7001).
user	User name for connecting to the running server to get MBean information (for example, weblogic).
password	Password for the user name.
compositeName	Name of the SOA composite application.
revision	Revision of the SOA composite application.
label	Optional. Label of the SOA composite application. The label identifies the MDS artifacts associated with the application. If the label is not specified, the system finds the latest one.

Example 43–8 Activate an Application

```
ant -f ant-sca-mgmt.xml activateComposite -Dhost=myhost -Dport=8001
```

```
-Duser=weblogic-DcompositeName=HelloWorld -Drevision=1.0
```

Note: After specifying the user name, enter the password when prompted.

Argument	Definition
host	Hostname of the Oracle WebLogic Server (for example, myhost).
port	Port of the Oracle WebLogic Server (for example, 7001).
user	User name for connecting to the running server to get MBean information (for example, weblogic).
password	Password for the user name.
compositeName	Name of the SOA composite application.
revision	Revision of the SOA composite application.
label	Optional. Label of the SOA composite application. The label identifies the MDS artifacts associated with the application. If the label is not specified, the system finds the latest one.

Example 43–9 Retire an Application

```
ant -f ant-sca-mgmt.xml retireComposite -Dhost=myhost -Dport=8001 -Duser=weblogic -DcompositeName=HelloWorld -Drevision=1.0
```

Note: After specifying the user name, enter the password when prompted.

Argument	Definition
host	Hostname of the Oracle WebLogic Server (for example, myhost).
port	Port of the Oracle WebLogic Server (for example, 7001).
user	User name for connecting to the running server to get MBean information (for example, weblogic).
password	Password for the user name.
compositeName	Name of the SOA composite application.
revision	Revision of the SOA composite application.
label	Optional. Label of the SOA composite application. The label identifies the MDS artifacts associated with the application. If the label is not specified, the system finds the latest one.

Example 43–10 Assigning the Default Version to a SOA Composite Application

```
ant -f ant-sca-mgmt.xml assignDefaultComposite -Dhost=myhost -Dport=8001 -Duser=weblogic -DcompositeName=HelloWorld -Drevision=1.0
```

Note: After specifying the user name, enter the password when prompted.

Argument	Definition
host	Hostname of the Oracle WebLogic Server (for example, myhost).
port	Port of the Oracle WebLogic Server (for example, 7001).
user	User name for connecting to the running server to get MBean information (for example, weblogic).
password	Password for the user name.
compositeName	Name of the SOA composite application.
revision	Revision of the SOA composite application.

Example 43–11 Listing the Deployed SOA Composite Applications

```
ant -f ant-sca-mgmt.xml listDeployedComposites -Dhost=myhost -Dport=8001
-Duser=weblogic
```

Note: After specifying the user name, enter the password when prompted.

Argument	Definition
host	Hostname of the Oracle WebLogic Server (for example, myhost).
port	Port of the Oracle WebLogic Server (for example, 7001).
user	User name for connecting to the running server to get MBean information (for example, weblogic).
password	Password for the user name.

43.6.2.7 Upgrading a SOA Composite Application

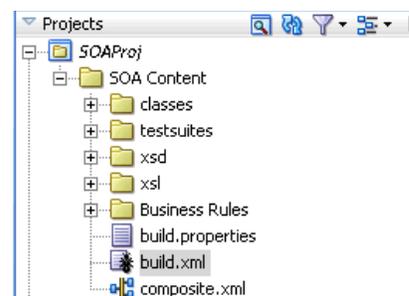
You can use `ant` to upgrade a SOA composite application from 10.1.3 to 11g. For information, see *Oracle Fusion Middleware Upgrade Guide for Oracle SOA Suite, WebCenter, and ADF*.

43.6.2.8 How to Manage SOA Composite Applications with ant Scripts

The WebLogic Fusion Order Demo application provides an example of using `ant` scripts to compile, package, and deploy the application. You can create the initial `ant` build files by selecting **New > Ant > Buildfile from Project** from the **File** main menu.

Figure 43–18 shows the `build.properties` and `build.xml` files that display in the Application Navigator after creation.

Figure 43–18 Ant Build Files



- **build.properties**

A file that you edit to reflect your environment (for example, specifying Oracle home and Java home directories, setting server properties such as hostname and port number to use for deployment, specifying the application to deploy, and so on).

- **build.xml**

Used by ant to compile, build, and deploy composite applications to the server specified in the **build.properties** file.

1. Modify the `build.properties` file to reflect your environment.

2. From the **Build** menu, select **Run Ant on *project_name***.

This builds targets defined in the current project's build file.

For more information about downloading the WebLogic Fusion Order Demo and modifying and using these files, see *Oracle Fusion Middleware Tutorial for Running and Building an Application with Oracle SOA Suite*.

43.7 Moving SOA Composite Applications to and from Development, Test, and Production Environments

As you move projects from one environment to another (for example, from testing to production), you typically must modify several environment-specific values, such as JDBC connection strings, hostnames of various servers, and so on. Configuration plans enable you to modify these values using a single text (XML) file called a configuration plan. The configuration plan is created in either Oracle JDeveloper or from the command line. During process deployment, the configuration plan is used to search the SOA project for values that must be replaced to adapt the project to the next target environment.

43.7.1 Introduction to Configuration Plans

A configuration plan enables you to replace the following attributes and properties:

- You create and edit a configuration plan file in which you can replace the following attributes and properties:
 - Any composite, service component, reference, service, and binding properties in the SOA composite application file (`composite.xml`)
 - Attribute values for bindings (for example, the location for `binding.ws`)
 - `schemaLocation` attribute of an import in a WSDL file
 - `location` attribute of an include in a WSDL file
 - `schemaLocation` attribute of an include, import, and redefine in an XSD file
 - Any properties in JCA adapter files
 - Modify and add policy references for the following:
 - * Service component
 - * Service and reference binding components

Note: The configuration plan does not alter XSLT artifacts in the SOA composite application. If you want to modify any XSL, do so in the XSLT Mapper. Using a configuration plan is not useful. For example, you cannot change references in XSL using the configuration plan file. Instead, they must be changed manually in the XSLT Mapper in Oracle JDeveloper when moving to and from test, development, and production environments. This ensures that the XSLT Mapper opens without any issues in design time. However, leaving the references unchanged does not impact runtime behavior.

- You attach the configuration plan file to a SOA composite application JAR file or ZIP file (if deploying a SOA bundle).
- During deployment, the configuration plan file is used to search the `composite.xml`, WSDL, and XSD files in the SOA composite application JAR file for values that must be replaced to adapt the project to the next target environment.

43.7.2 Introduction to a Configuration Plan File

The following example shows a configuration plan in which you modify the following:

- An `inFileFolder` property for composite `FileAdaptorComposite` is replaced with `mytestserver/newinFileFolder`.
- A hostname (`myserver17`) is replaced with `test-server` and port `8888` is replaced with `8198` in the following locations:
 - All import WSDLs
 - All reference `binding.ws` locations

The `composite.xml` file looks as shown in [Example 43–12](#):

Example 43–12 *composite.xml* File

```
<composite . . . . .>
  <import namespace="http://example.com/hr/"
    location="http://myserver17.us.oracle.com:8888/hrapp/HRAppService?WSDL"
    importType="wsdl" />
  <service name="readPO">
    <interface.wsdl
      interface="http://xmlns.oracle.com/pcbpel/adaptor/file/readPO/#wsdl.interface(Read
      _ptt)" />
    <binding.jca config="readPO_file.jca" />
    <property name="inFileFolder" type="xs:string" many="false"
      override="may">/tmp/inFile</property>
    </service>
  <reference name="HRApp">
    <interface.wsdl
      interface="http://example.com/hr/#wsdl.interface(HRAppService)" />
    <binding.ws
      port="http://example.com/hr/#wsdl.endpoint(HRAppService/HRAppServiceSoapHttpPort)"
      location="http://myserver17.us.oracle.com:8888/hrapp/HRAppService?WSDL" />
    <binding.java serviceName="{http://example.com/hr/}HRAppService"
      registryName="HRAppCodeGen_JBOServiceRegistry" />
    </reference>
  </composite>
```

The configuration plan file looks as shown in [Example 43–13](#).

Example 43–13 Configuration Plan File

```
<?xml version="1.0" encoding="UTF-8"?>
<soaConfigPlan xmlns:jca="http://xmlns.oracle.com/pcbpel/wsdl/jca/">
  <composite name="FileAdaptorComposite">
    <service name="readPO">
      <binding type="*">
        <property name="inFileFolder">
          <replace>/mytestserver/newinFileFolder</replace>
        </property>
      </binding>
    </service>
  </composite>
  <!-- For all composite replace host and port in all imports wsdl -->
  <composite name="*">
    <imports>
      <searchReplace>
        <search>myserver17</search>
        <replace>test-server</replace>
      </searchReplace>
      <searchReplace>
        <search>8888</search>
        <replace>8198</replace>
      </searchReplace>
    </imports>
    <reference name="*">
      <binding type="ws">
        <attribute name="location">
          <searchReplace>
            <search>myserver17</search>
            <replace>test-server</replace>
          </searchReplace>
          <searchReplace>
            <search>8888</search>
            <replace>8198</replace>
          </searchReplace>
        </attribute>
      </binding>
    </reference>
  </composite>
</soaConfigPlan>
```

A policy is replaced if a policy for the same URI is available. Otherwise, it is added. This is different from properties, which are modified, but not added.

43.7.3 Introduction to Use Cases for a Configuration Plan

The following steps provide an overview of how to use a configuration plan when moving from development to testing environments:

1. User A creates SOA composite application Foo.
2. User A deploys Foo to a development server, fixes bugs, and refines the process until it is ready to test in the staging area.
3. User A creates and edits a configuration plan for Foo, which enables the URLs and properties in the application to be modified to match the testing environment.

4. User A deploys Foo to the testing server using Oracle JDeveloper or a series of command-line scripts (can be WLST-based). The configuration plan created in Step 3 modifies the URLs and properties in Foo.
5. User A deploys SOA composite application Bar in the future and applies the same plan during deployment. The URLs and properties are also modified.

The following steps provide an overview of how to use a configuration plan when creating environment-independent processes:

Note: This use case is useful for users that have their own development server and a common development and testing server if they share development of the same process. Users that share the same deployment environment (that is, the same development server) may not find this use case as useful.

1. User A creates SOA composite application Foo.
2. User A deploys Foo to their development server, fixes bugs, and refines the process until it is ready to test in the staging area.
3. User A creates a configuration plan for Foo, which enables the URLs and properties in the process to be modified to match the settings for User A's environment.
4. User A checks in Foo and the configuration plan created in Step 3 to a source control system.
5. User B checks out Foo from source control.
6. User B makes a copy of the configuration plan to match their environment and applies the new configuration plan onto Foo's artifacts.
7. User B imports the application into Oracle JDeveloper and makes several changes.
8. User B checks in both Foo and configuration plan B (which matches user B's environment).
9. User A checks out Foo again, along with both configuration plans.

43.7.4 How to Create a Configuration Plan in Oracle JDeveloper

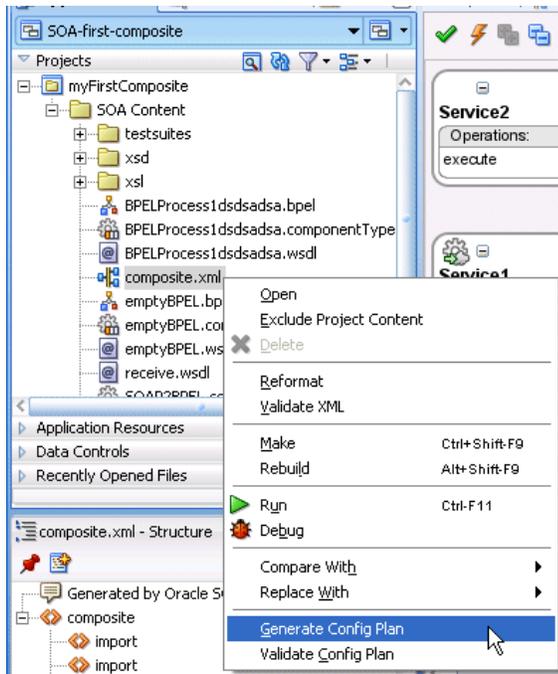
This section describes how to create and use a configuration plan. In particular, this section describes the following:

- Creating and editing a configuration plan
- Attaching the configuration plan to a SOA composite application JAR file
- Validating the configuration plan
- Deploying the SOA composite application JAR or ZIP file in which the configuration plan is included

To create a configuration plan in Oracle JDeveloper:

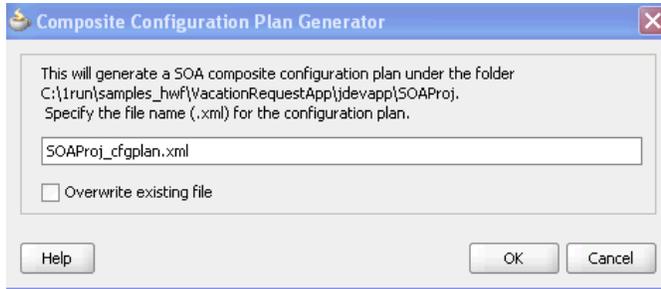
1. Open Oracle JDeveloper.
2. Right-click the **composite.xml** file of the project in which to create a configuration plan, and select **Generate Config Plan**. [Figure 43–19](#) provides details.

Figure 43–19 Generate a Configuration Plan



The Composite Configuration Plan Generator dialog appears.

Figure 43–20 Composite Configuration Plan Generator Dialog



3. Create a configuration plan file for editing, as shown in [Table 43–6](#).

Table 43–6 Generate a Configuration Plan

Field	Description
Specify the file name (.xml) for the configuration plan	Enter a specific name or accept the default name for the configuration plan. The file is created in the directory of the project and packaged with the SOA composite application JAR or ZIP file. Note: During deployment, you can specify a different configuration file when prompted in the SOA Deployment Configuration dialog.
Overwrite existing file	Click to overwrite an existing configuration plan file with a different file in the project directory.

4. Click **OK**.

This creates and opens a single configuration plan file for editing, similar to that shown in [Example 43-13](#) on page 43-32. You can modify URLs and properties for the `composite.xml`, WSDL, and schema files of the SOA composite application. [Figure 43-21](#) provides details.

Figure 43-21 Configuration Plan Editor

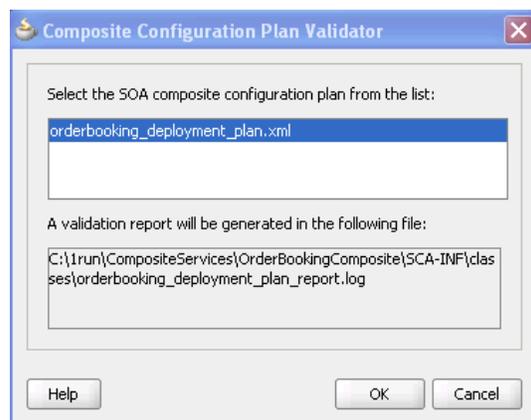


```
<?xml version="1.0" encoding="UTF-8"?>
<SOAConfigPlan xmlns:jca="http://platform.integration.oracle/blocks/adap
<composite name="Project1">
  <!--Add search and replace rules for the import section of a compo
  Example:
  <searchReplace>
    <search>http://my-dev-server</search>
    <replace>http://my-test-server</replace>
  </searchReplace>
  <searchReplace>
    <search>8888</search>
    <replace>8889</replace>
  </searchReplace-->
  <import>
    <searchReplace>
      <search/>
      <replace/>
    </searchReplace>
  </import>
  <service name="bpelprocess1_client_ep">
    <binding type="ws">
      <attribute name="port">
        <replace>http://xmlns.oracle.com/MyApp/Project1/BPELProce
      </attribute>
    </binding>
  </service>
  <!--Add search and replace rules for the component properties
  For components and service/reference bindings, you can add polic
```

5. Add values for server names, port numbers, and so on to the existing syntax. You can also add replacement-only syntax when providing a new value. You can add multiple search and replacement commands in each section.
6. From the **File** menu, select **Save All**.
7. Above the editor, click the **x** to the right of the file name to close the configuration plan file.
8. Right-click the `composite.xml` file again, and select **Validate Config Plan**.

The Composite Configuration Plan Validator appears, as shown in [Figure 43-22](#).

Figure 43-22 Validate the Configuration Plan



9. Select the configuration plan to validate. This step identifies all search and replacement changes to be made during deployment. Use this option for debugging only.
10. Note the directory in which a report describing validation results is created, and click **OK**.

The Log window in Oracle JDeveloper indicates if validation succeeded and lists all search and replacement commands to perform during SOA composite application deployment. This information is also written to the validation report.

Note: The old `composite.xml`, WSDL, and XSD files are not replaced with files containing the new values for the URLs and properties appropriate to the next environment. Replacement occurs only when the SOA composite application is deployed.

11. Deploy the SOA composite application by following the instructions in one of the following sections:
 - [Section 43.2, "Deploying a Single SOA Composite in Oracle JDeveloper"](#)
 - [Section 43.3, "Deploying Multiple SOA Composite Applications in Oracle JDeveloper"](#)
 - [Section 43.4, "Deploying and Using Shared Metadata Across SOA Composite Applications"](#)

During deployment, the SOA Deployment Configuration dialog shown in Step 5 on page 43-5 prompts you to select the configuration plan to include in the SOA composite application archive.

12. Select the configuration plan to include with the SOA composite application.
13. Click **OK**.

43.7.5 How to Create a Configuration Plan with the WLST Utility

As an alternative to using Oracle JDeveloper, you can use the WLST command line utility to perform the following configuration plan management tasks:

- Generate a configuration plan for editing
- Attach the configuration plan file to the SOA composite application JAR file
- Validate the configuration plan
- Extract a configuration plan packaged with the JAR file for editing

For instructions, see *Oracle Fusion Middleware WebLogic Scripting Tool Command Reference*.

Using Business Events and the Event Delivery Network

This chapter describes how to publish and subscribe to business events in a SOA composite application. Business events consist of message data sent as the result of an occurrence in a business environment. When a business event is published, other service components can subscribe to it.

This chapter includes the following sections:

- [Section 44.1, "Introduction to Business Events"](#)
- [Section 44.2, "Creating Business Events in Oracle JDeveloper"](#)

For samples that show how to use business events with Oracle Mediator, visit the following URL:

http://www.oracle.com/technology/sample_code/products/mediator

For additional information on creating business events in a SOA composite application, see *Oracle Fusion Middleware Tutorial for Running and Building an Application with Oracle SOA Suite*.

44.1 Introduction to Business Events

You can raise business events when a situation of interest occurs. For example, in a loan flow scenario, a BPEL process executing a loan process can raise a *loan completed event* at the completion of the process. Other systems within the infrastructure of this application can listen for these events and, upon receipt of one instance of an event:

- Use the event context to derive business intelligence or dashboard data.
- Signal to a mail department that a loan package must be sent to a customer.
- Invoke another business process.
- Send information to Oracle Business Activity Monitoring (BAM)

Business events are typically a one-way, fire-and-forget, asynchronous way to send a notification of a business occurrence. The business process does *not*:

- Rely on any service component receiving the business event to complete.
- Care if any other service components receive the business event.
- Need to know where subscribers (if any) are and what they do with the data.

These are important distinctions between business events and direct service invocations that rely on the Web Services Description Language (WSDL) file contract (for example, a SOAP service client). If the author of the event depends on the receiver

of the event, then messaging typically must be accomplished through service invocation rather than through a business event. Unlike direct service invocation, the business event separates the client from the server.

A business event is defined using the event definition language (EDL). EDL is a schema used to build business event definitions. Applications work with instances of the business event definition.

EDL consists of the following:

- **Global name**
Typically a Java package name (for example, `com.acme.ExpenseReport.created`), though this is not required.
- **Custom headers**
Used for fast routing. For example, if an event named Expense Report Created has a Currency header, the component that creates the event at runtime is responsible for populating the Currency header. The event can be routed based on the value of the header more quickly than doing an XPath query into the event payload.
- **Payload definition**
The most common use for a definition is an XML Schema (XSD). The payload of a business event is defined using an XSD. The schema URI is contained in the root element of the payload.

[Example 44-1](#) shows an EDL file with two business events in the `BugReport` event definition: `bugUpdated` and `bugCreated`. The namespace (`BugReport`) and associated schema file (`BugReport.xsd`) are referenced.

Example 44-1 EDL File with Two Business Events

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<definitions targetNamespace="/model/events/edl/BugReport"
  xmlns:ns0="/model/events/schema/BugReport"
  xmlns="http://schemas.oracle.com/events/edl">
  <schema-import namespace="/model/events/schema/BugReport"
    location="BugReport.xsd" />

  <event-definition name="bugCreated">
    <content element="ns0:bugCreatedInfo"/>
  </event-definition>

  <event-definition name="bugUpdated">
    <content element="ns0:bugUpdatedInfo"/>
  </event-definition>
</definitions>
```

These two events are available for subscription in Oracle Mediator.

Business events are deployed to the metadata service (MDS) repository. Deploying a business event to MDS along with its artifacts (for example, the XSDs) is known as publishing the EDL (or event definition). This action transfers the EDL and its artifacts to a shared area in MDS. An object in an MDS shared area is visible to all applications in the Resource Palette of Oracle JDeveloper. After an EDL is published, it can be subscribed to by other applications. EDLs cannot be unpublished; the definition always exists.

A subscription is for a specific qualified name (QName) (for example, `x.y.z/newOrders`). A QName is a tuple (URI, localName) that may be derived

from a string `prefix:localName` with a namespace declaration such as `xmlns:prefix=URI` or a namespace context. In addition, subscriptions can be further narrowed down by using content-based filters.

Business events are published in the Event Delivery Network (EDN). The EDN runs within every SOA instance. Raised events are delivered by EDN to the subscribing service components.

For this release, the following SOA service components and actions are supported:

- Oracle Mediator can subscribe to and publish events.
- Oracle Mediator can subscribe to events that are then wired to the BPEL process. This feature enables you to pass the principal through the security interceptor for component-level authorization policy enforcement.

Notes: There are two implementations of the EDN: JMS and AQ (provides support for PL/SQL APIs).

44.1.1 Local and Remote Events Boundaries

A single SOA composite application instance can reside in a single container or can be clustered across multiple containers. Another application (for example, an Oracle Application Development Framework (ADF) Business Component application) can be configured to run in the same container as the SOA composite application instance or in a different container.

Raising an event from a Java EE application can be done through a local event connection or a remote event connection:

- Local event connection

If the publisher resides on the same Oracle WebLogic Server as the application and the publisher uses a local business event connection factory, the event is raised through a local event connection. In this scenario, synchronous subscriptions are executed synchronously.
- Remote event connection

If the caller resides in a different container (different JVM) then the application, the event is raised through a remote event connection. Only asynchronous subscriptions are supported for remote event connections.

You can also raise events through PL/SQL APIs.

If another application (for example, an Oracle ADF Business Component application) is configured to run in the same container as the SOA composite application, it is optimized to use local event connections. The boundary for events is the application instance. When an event is raised in the application instance, subscriptions registered in the application instance are executed. Events are not propagated from one application instance to another. Propagation can be achieved through an Oracle Mediator in both instances, which listens to events and publishes them to a JMS queue.

44.2 Creating Business Events in Oracle JDeveloper

This section provides a high-level overview of how to create and subscribe to a business event. In this scenario, a business event named **NewOrderSubmitted** is created when a user places an order in a store front application (**StoreFrontService** service). You then create an Oracle Mediator service component to subscribe to this business event.

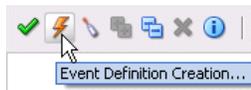
This scenario is part of the larger WebLogic Fusion Order Demo application. For additional details about this application, see the *Oracle Fusion Middleware Tutorial for Running and Building an Application with Oracle SOA Suite*.

44.2.1 How to Create a Business Event

To create a business event:

1. Create a SOA project as an empty composite.
2. Launch the Event Definition Creation wizard in either of two ways:
 - a. In the SOA Composite Editor, click the icon above the designer. [Figure 44–1](#) provides an example.

Figure 44–1 Event Definition Creation



- b. From the **File** main menu, select **New > SOA Tier > Service Components > Event Definition**.

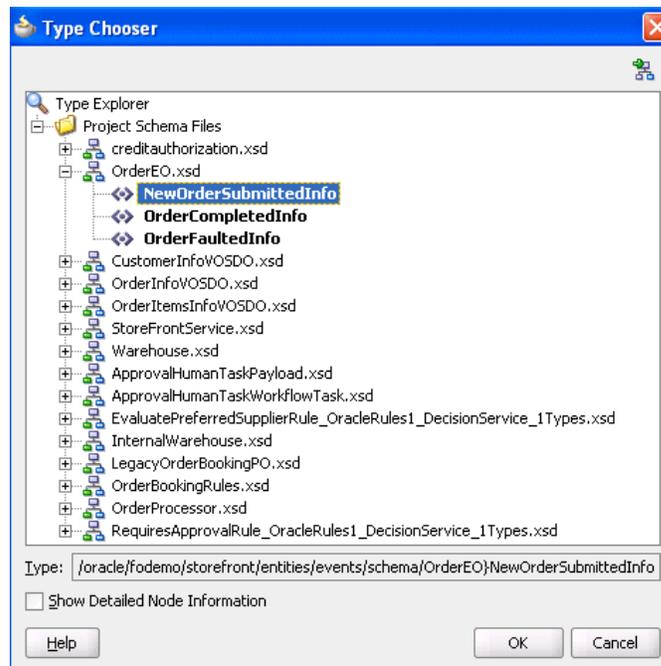
The Event Definition Creation dialog appears.

3. Enter the details described in [Table 44–1](#).

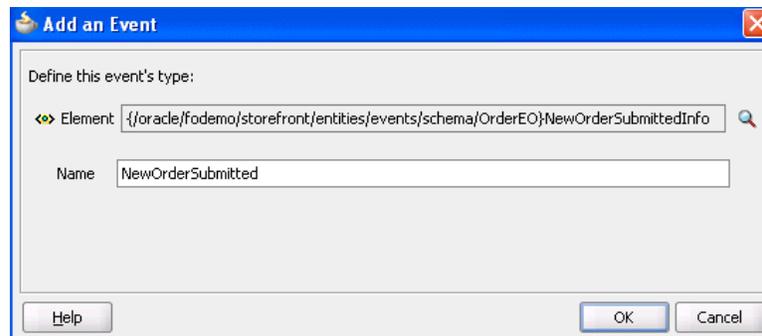
Table 44–1 Event Definition Creation Wizard Fields and Values

Field	Value
Event Definition Name	Enter a name. Note: Do not enter a forward slash (/) as the event name. This creates an event definition file consisting of only an extension for a name (.edn).
Directory	Displays the directory path.
Namespace	Accept the default namespace or enter a value for the namespace in which to place the event.

4. Click the **Add** icon to add an event.
The Add an Event dialog appears.
5. Click the **Search** icon to select the payload, and click **OK**. [Figure 44–2](#) provides details.

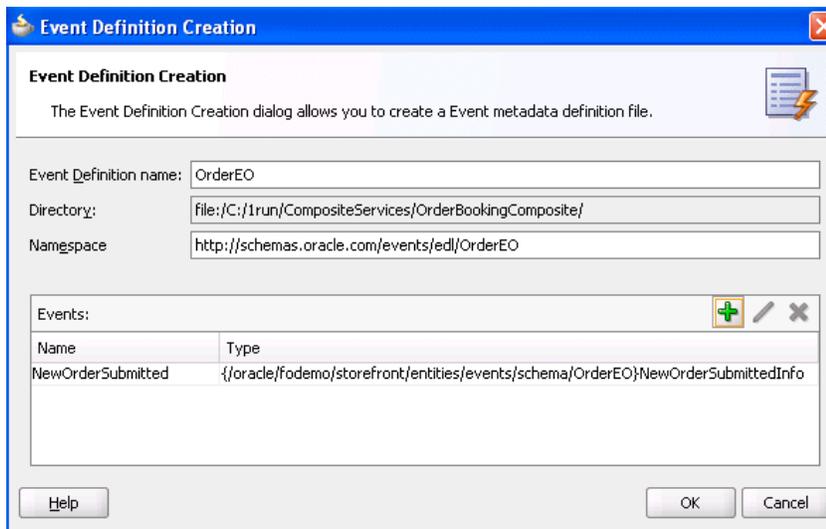
Figure 44–2 Select the Payload

6. In the **Name** field, enter a name, as shown in [Figure 44–3](#).

Figure 44–3 Add an Event Dialog

7. Click **OK**.

The added event now appears in the **Events** section, as shown in [Figure 44–4](#).

Figure 44–4 Event Definition Creation Dialog

8. Above the editor, click the x next to *event_definition_name.edl* to close the Events editor.
9. Click **Yes** when prompted to save your changes. If you do not save your changes, the event is not created and cannot be selected in the Event Chooser window.

The business event is published to MDS and you are returned to the SOA Composite Editor. The business event displays for browsing in the Resource Palette.

44.2.2 How to Subscribe to a Business Event

To subscribe to a business event:

1. From the Component Palette, drag a **Mediator** service component into the SOA Composite Editor. This service component enables you to subscribe to the business event.
2. In the **Name** field, enter a name.
3. From the **Options** list, select **Subscribe to Event**.

The window is refreshed to display an events table.

4. Click the **Add** icon to select an event.
The Event Chooser window appears.
5. Select the event you created and click **OK**.
You are returned to the Create Mediator dialog.
6. Select a level of delivery consistency for the event.

- **one and only one**

Events are delivered to the subscriber in its own global (that is, JTA) transaction. Any changes made by the subscriber within that transaction are committed after the event processing is complete. If the subscriber fails, the transaction is rolled back. Failed events are retried a configured number of times.

- **guaranteed**

Events are delivered to the subscriber asynchronously without a global transaction. The subscriber can choose to create its own local transaction for processing, but it is committed independently of the rest of the event processing. The event is guaranteed to be handed to the subscriber, but because there is no global transaction, there is a possibility that a system failure can cause an event to be delivered more than once. If the subscriber throws an exception (or fails in any way), the exception is logged, but the event is not resent.

- **immediate**

Events are delivered to the subscriber in the same global transaction and same thread as the publisher. The publish call does not return until all immediate subscribers have completed processing. If any subscribers throw an exception, no additional subscribers are invoked and an exception is thrown to the publisher. The transaction is rolled back in case of any error during immediate processing.

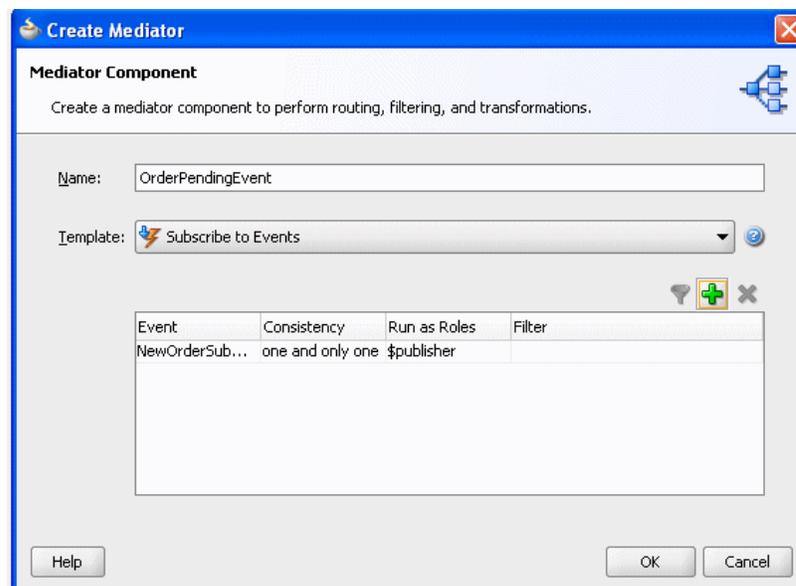
7. If you want to filter the event, double-click the **Filter** column of the selected event or select the event and click the **filter** icon (first icon) above the table. This displays the Expression Builder dialog. This dialog enables you to specify an XPath filter expression. A filter expression specifies that the contents (payload or headers) of a message be analyzed before any service is invoked. For example, you can apply a filter expression that specifies that a service be invoked only if the message includes a customer ID.

When the expression logic is satisfied, the event is accepted for delivery.

For more information about filters, see [Section 19.2.2.6, "Specifying Expression for Filtering Messages."](#)

Figure 44–5 shows the Create Mediator dialog.

Figure 44–5 Create Mediator Dialog



8. Click **OK**.

Figure 44–6 shows an icon on the left side that indicates that Oracle Mediator is configured for an event subscription.

Figure 44–6 Configuration for Event Subscription



44.2.3 What Happens When You Create and Subscribe to a Business Event

The source code in Example 44–2 provides details about the subscribed event of the Oracle Mediator service component.

Example 44–2 Subscribed Event

```
<component name="OrderPendingEvent">
  <implementation.mediator src="OrderPendingEvent.mplan"/>
  <business-events>
    <subscribe
      xmlns:sub1="/oracle/fodemo/storefront/entities/events/edl/OrderEO"
      name="sub1:NewOrderSubmitted" consistency="oneAndOnlyOne"
      runAsRoles="$publisher"/>
  </business-events>
</component>
```

While not explicitly demonstrated in this example, you can define XPath filters on events. In Example 44–3, the event is accepted for delivery *only* if the initial deposit is greater than 50000:

Example 44–3 Definition of XPath Filters on Events

```
<business-events>
  . . .
  . . .
  <filter>
    <xpath xmlns:be="http://oracle.com/fabric/businessEvent"
      xmlns:ns1="http://xmlns.oracle.com/singleString"
      <xpath expression= "/be:business-event/be:content/
        sub1:AccountInfo/Details[@initialDeposit > 50000]" />
    </filter>
  . . .
  . . .
</business-events>
```

44.2.4 What You May Need to Know About Subscribing to a Business Event

Note that subscribers in nondefault revisions of SOA composite applications can still get business events. For example, note the following behavior:

1. Create a composite application with an initial Oracle Mediator service component named M1 that publishes an event and a second Oracle Mediator service component named M2 that subscribes to the event. The output is written to a directory.
2. Deploy the composite application as revision 1.

3. Modify the composite application by adding a third Oracle Mediator service component named M3 that subscribes to the same event and writes the output to a different directory.
4. Deploy the composite application as revision 2 (the default).
5. Invoke revision 2 of the composite application.

Note that Oracle Mediator M2 writes the output to two files with the same content in the directory. As expected, Oracle Mediator M3 picks up the event and writes the output successfully to another directory. However, note that Oracle Mediator M2 (from revision 1) is also picking up and processing the published event from revision 2 of the composite application. Therefore, it creates one more output file in the same directory.

44.2.5 How to Publish a Business Event

You can create a second Oracle Mediator to publish the event that you subscribed to in [Section 44.2.2, "How to Subscribe to a Business Event."](#)

To publish a business event:

1. Create a second Oracle Mediator service component that publishes the event to which the first Oracle Mediator subscribes.
2. Return to the first Oracle Mediator service component.
3. In the **Routing Rules** section, click the **Add** icon.
4. Click **Service** when prompted by the Target Type window.
5. Select the second Oracle Mediator service component.
6. Select **Save All** from the **File** main menu.

44.2.6 What Happens When You Publish a Business Event

Note that the two Oracle Mediator service components appear in [Example 44–4](#). One service component (`OrderPendingEvent`) subscribes to the event and the other service component (`PublishOrderPendingEvent`) publishes the event.

Example 44–4 Event Subscription and Publication

```
<component name="PublishOrderPendingEvent">
  <implementation.mediator src="PublishOrderPendingEvent.mplan"/>
  <business-events>
    <publishes
      xmlns:sub1="/oracle/fodemo/storefront/entities/events/ed1/OrderEO"
      name="pub1:NewOrderSubmitted"/>
    </business-events>
  </component>

<component name="OrderPendingEvent">
  <implementation.mediator src="OrderPendingEvent.mplan"/>
  <business-events>
    <subscribe
      xmlns:sub1="/oracle/fodemo/storefront/entities/events/ed1/OrderEO"
      name="sub1:NewOrderSubmitted" consistency="oneAndOnlyOne"
      runAsRoles="$publisher"/>
    </business-events>
  </component>
```

44.2.7 How to Integrate Oracle ADF Business Component Business Events with Oracle Mediator

This section provides a high-level overview of how to integrate Oracle ADF Business Component event conditions with Oracle Mediator.

To integrate Oracle ADF Business Component business events with Oracle Mediator:

1. Create a business component project.
2. Add a business event definition to the project. This action generates an EDL file and an XSD file. The XSD file contains the definition of the payload. Ensure also that you specify that the event be raised by the Oracle ADF Business Component upon creation.
3. Create an SOA composite application and manually copy the EDL and XSD schema files to the SOA project's root directory. For example:

```
JDeveloper/mywork/SOA_application_name/SOA_project_name
```
4. Place schema files at the proper relative location from the EDL file based on the import.
5. Create an Oracle Mediator service component as described in [Section 44.2.2, "How to Subscribe to a Business Event."](#)
6. In the Event Chooser window, select the event's EDL file, as described in [Section 44.2.2, "How to Subscribe to a Business Event."](#)
7. Create a BPEL process service component in the same SOA composite application for the Oracle Mediator to invoke. In the **Input Element** field of the **Advanced** tab, ensure that you select the payload of the Business Component business event XSD created in Step 2.
8. Double-click the BPEL process.
9. Drag an **Email** activity into the BPEL process.
10. Use the payload of the business event XSD to complete the **Subject** and **Body** fields.
11. Return to the Oracle Mediator service component in the SOA Composite Editor.
12. Design a second service component to publish the event, such as a BPEL process or a second Oracle Mediator service component.

SOA composite application design is now complete.

For more information about creating and publishing Oracle ADF Business Component business events, see *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

Creating Transformations with the XSLT Mapper

This chapter describes how to use the XSLT Mapper. The XSLT Mapper enables you to create data transformations between source schema elements and target schema elements in either Oracle BPEL Process Manager or Oracle Mediator. Version 1.0 of XSLT is supported.

This chapter includes the following sections:

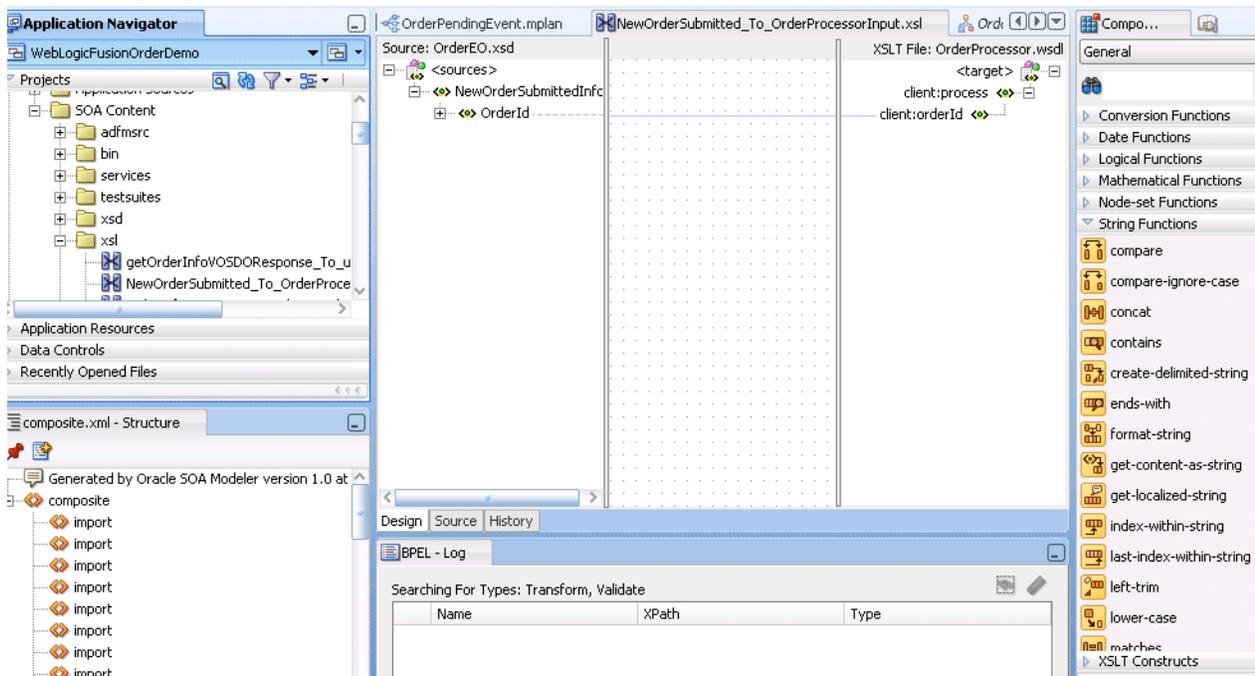
- [Section 45.1, "Introduction to the XSLT Mapper"](#)
- [Section 45.2, "Creating an XSL Map File"](#)
- [Section 45.3, "Designing Transformation Maps with the XSLT Mapper"](#)
- [Section 45.4, "Testing the Map"](#)
- [Section 45.5, "Demonstrating the New Features of the XSLT Mapper"](#)

For information on invoking the XSLT Mapper from Oracle BPEL Process Manager, see [Section 45.2.1, "How to Create an XSL Map File in Oracle BPEL Process Manager."](#) For information on invoking the XSLT Mapper from Oracle Mediator, see [Section 45.2.3, "How to Create an XSL Map File in Oracle Mediator."](#)

45.1 Introduction to the XSLT Mapper

You use the XSLT Mapper transformation tool to create the contents of a map file. [Figure 45-1](#) shows the layout of the XSLT Mapper.

Figure 45–1 Layout of the XSLT Mapper



The **Source** and the **Target** schemas are represented as trees and the nodes in the trees are represented using a variety of icons. The displayed icon reflects the schema or property of the node. For example:

- An XSD attribute is denoted with an icon that is different from an XSD element.
- An optional element is represented with an icon that is different from a mandatory element.
- A repeating element is represented with an icon that is different from a nonrepeating element, and so on.

The various properties of the element and attribute are displayed in the Property Inspector in the lower right of the XSLT Mapper when the element or attribute is selected. (for example, type, cardinality, and so on). The Component Palette in the upper right of Figure 45–1 is the container for all functions provided by the XSLT Mapper. The XSLT Mapper is the actual drawing area for dropping functions and connecting them to source and target nodes.

When an XSLT map is first created, the target tree shows the element and attribute structure of the target XSD. An XSLT map is created by inserting XSLT constructs and XPath expressions into the target tree at appropriate positions. When executed, the XSLT map generates the appropriate elements and attributes in the target XSD.

Editing can be done in design view or source view. When a map is first created, you are in design view. Design view provides a graphical display and enables editing of the map. To see the text representation of the XSLT being created, switch to source view. To switch views, click the **Source** or **Design** tabs at the bottom of the XSLT Mapper.

While in design view, the following pages from the Component Palette can be used:

- **General:** Commonly used XPath functions and XSLT constructs.
- **Advanced:** More advanced XPath functions such as database and cross-reference functions.

- **User Defined:** User-defined functions and templates. This page is visible only when the user has templates in their XSL or user-defined external functions defined through the preferences pages.
- **All Pages:** Provides a view of all functions in one page.
- **My Components:** Contains user favorites and recently-used functions. To add a function to your favorites, right-click the function in the Component Palette and select **Add to Favorites**.

While in source view, the XML and the <http://www.w3.org/1999/XSL/Transform> pages can be used.

The XSLT Mapper provides three separate context sensitive menus:

- One in the source panel
- One in the target panel
- One in the center panel

Right-click each of the three separate panels to see what the context menus look like.

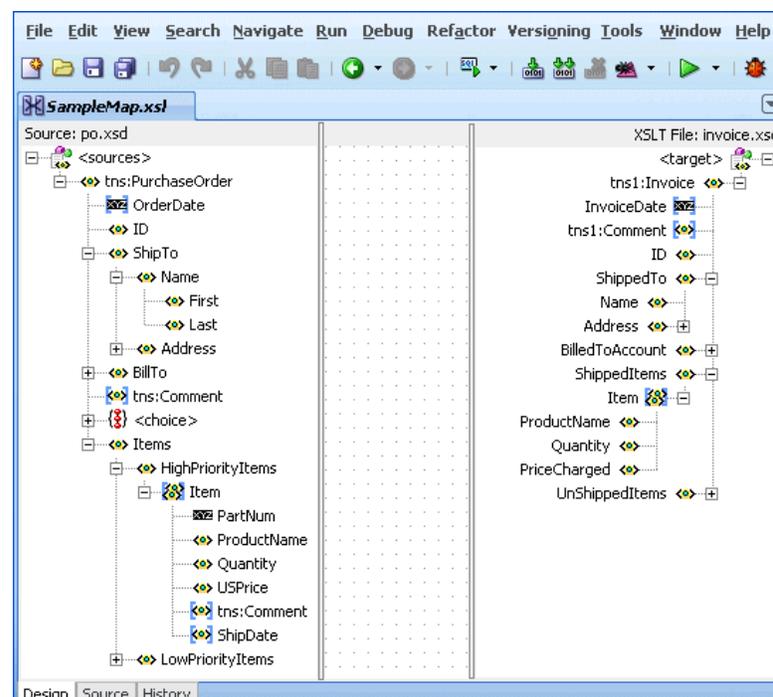
By default, design view shows all defined prefixes for all nodes in the source and target trees. You can elect not to display prefixes by selecting **Hide Prefixes** from the context menu in the center panel of the design view. After prefixes are hidden, select **Show Prefixes** to display them again.

45.1.1 Overview of XSLT Creation

It is important to understand how design view representation of the map relates to the generated XSLT in source view. This section provides a brief example.

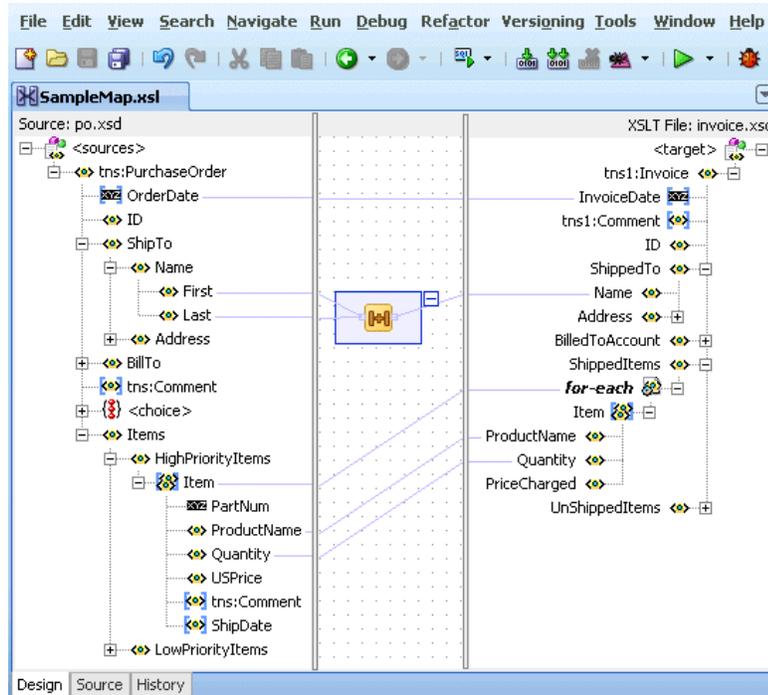
After creating an initial map, the XSLT Mapper displays a graphical representation of the source and target schemas, as shown in [Figure 45-2](#).

Figure 45-2 Source and Target Schemas



At this point, no target fields are mapped. Switching to source view displays an empty XSLT map. XSLT statements are built graphically in design view, and XSLT text is then generated. For example, design view mapping is shown in [Figure 45-3](#).

Figure 45-3 Design View Mapping



The design view results in the generation of the following XSLT statements in source view:

- The **OrderDate** attribute from the source tree is linked with a line to the **InvoiceDate** attribute in the target tree in [Figure 45-3](#). This results in a **value-of** statement in the XSLT, as shown in [Example 45-1](#).

Example 45-1 value-of Statement

```
<xsl:attribute name="InvoiceDate">
  <xsl:value-of select="/ns0:PurchaseOrder/@OrderDate"/>
</xsl:attribute>
```

- The **First** and **Last** name fields from the source tree in [Figure 45-3](#) are concatenated using an XPath **concat** function. The result is linked to the **Name** field in the target tree. This results in the XSLT statement shown in [Example 45-2](#):

Example 45-2 concat Function

```
<Name>
  <xsl:value-of select="concat (/ns0:PurchaseOrder/ShipTo/Name/First,
    /ns0:PurchaseOrder/ShipTo/Name/Last)"/>
</Name>
```

- Note the inserted XSLT **for-each** construct in the target tree in [Figure 45-3](#). For each **HighPriorityItems/Item** in the source tree, a **ShippedItems/Item** element is created in the target tree and **ProductName** and **Quantity** are copied for each. The XSLT shown in [Example 45-3](#) is generated:

Example 45–3 for-each Construct

```

<xsl:for-each
  select="/ns0:PurchaseOrder/Items/HighPriorityItems/Item">
  <Item>
    <ProductName>
      <xsl:value-of select="ProductName"/>
    </ProductName>
    <Quantity>
      <xsl:value-of select="Quantity"/>
    </Quantity>
  </Item>
</xsl:for-each>

```

The line linking **Item** in the source tree to the **for-each** construct in the target tree in [Figure 45–3](#) determines the XPath expression used in the **for-each** select attribute. In general, XSLT constructs have a select or test attribute that is populated by an XPath statement typically referencing a source tree element.

Note that the XPath expressions in the **value-of** statements beneath the **for-each** construct are relative to the XPath referenced in the **for-each**. In general, the XSLT Mapper creates relative paths within **for-each** statements.

If you must create an absolute path within a **for-each** construct, you must do this within source view. When switching back to design view, it is remembered that the path is absolute and the XSLT Mapper does not modify it.

Note: In [Example 45–3](#), the fields `ProductName` and `Quantity` are required fields in both the source and target. If these fields are optional in the source and target, it is a good practice to insert an `xsl:if` statement around these mappings to test for the existence of the source node. If this is not done, and the source node does not exist in the input document, an empty node is created in the target document. For example, if `ProductName` is optional in both the source and target, map them as follows:

```

<xsl:if test="ProductName">
  <ProductName>
    <xsl:value-of select="ProductName"/>
  </ProductName>
</xsl:if>

```

The entire XSLT map generated for this example is shown in [Example 45–4](#):

Example 45–4 Entire XSLT Map

```

<xsl:template match="/">
  <tnsl:Invoice>
    <xsl:attribute name="InvoiceDate">
      <xsl:value-of select="/ns0:PurchaseOrder/@OrderDate"/>
    </xsl:attribute>
    <ShippedTo>
      <Name>
        <xsl:value-of select="concat
(/ns0:PurchaseOrder/ShipTo/Name/First,/ns0:PurchaseOrder/ShipTo/Name/Last)"/>
      </Name>
    </ShippedTo>
    <ShippedItems>
      <xsl:for-each select="/ns0:PurchaseOrder/Items/HighPriorityItems/Item">

```

```

    <Item>
      <ProductName>
        <xsl:value-of select="ProductName" />
      </ProductName>
      <Quantity>
        <xsl:value-of select="Quantity" />
      </Quantity>
    </Item>
  </xsl:for-each>
</ShippedItems>
</tnsl:Invoice>
</xsl:template>

```

Subsequent sections of this chapter describe how to link source and target elements, add XSLT constructs, and create XPath expressions in design view.

45.1.2 Guidelines for Using the XSLT Mapper

- A node in the target tree can be linked only once (that is, you cannot have two links connecting a node in the target tree).
- An incomplete function and expression does not result in an XPath expression in source view. If you switch from design view to source view with one or more incomplete expressions, the Mapper Messages window displays warning messages.
- When you map duplicate elements in the XSLT Mapper, the style sheet becomes invalid and you cannot work in the **Design** view. The Log window shows the following error messages when you map an element with a duplicate name:

```

Error: This Node is Already Mapped :
"/ns0:rulebase/for-each/ns0:if/ns0:atom/ns0:rel"
Error: This Node is Already Mapped :
"/ns0:rulebase/for-each/ns0:if/ns0:atom/choice_1/ns0:ind"
Error: This Node is Already Mapped :
"/ns0:rulebase/for-each/ns0:if/ns0:atom/choice_1/ns0:var"

```

Duplicate nodes can be created in design view by surrounding each duplicate node with a **for-each** statement that executes once.

45.2 Creating an XSL Map File

Transformations are performed in an XSL map file in which you map source schema elements to target schema elements. This section describes methods for creating the XSL map file.

Note: You can also create an XSL map file from an XSL style sheet. Click **New > General > XML > XSL Map From XSL Stylesheet** from the **File** main menu in Oracle JDeveloper.

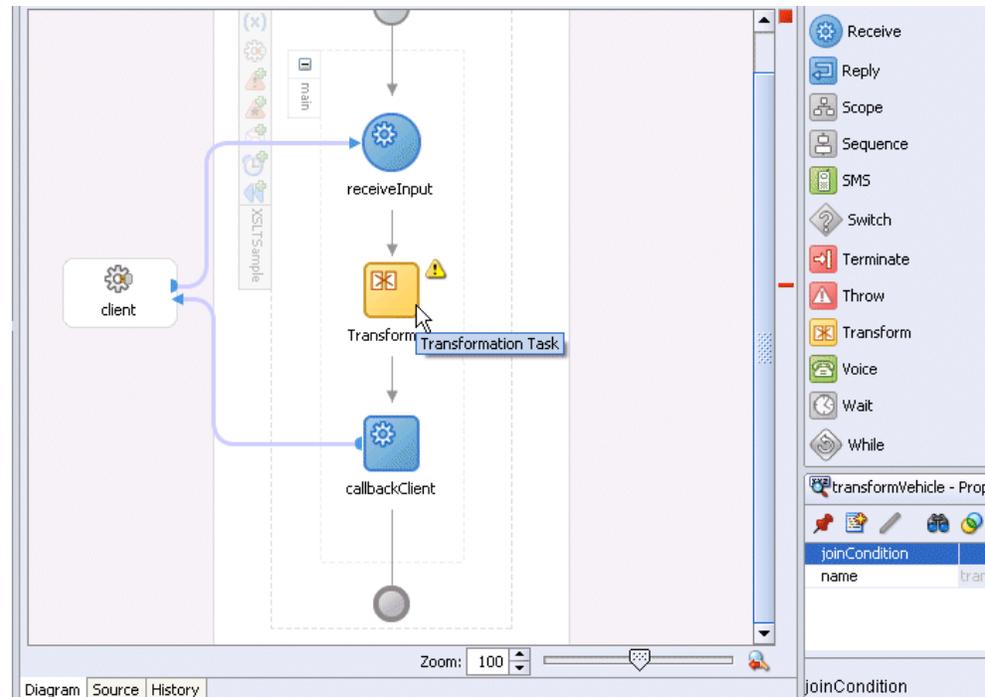
45.2.1 How to Create an XSL Map File in Oracle BPEL Process Manager

A transform activity enables you to create a transformation using the XSLT Mapper in Oracle BPEL Process Manager. This tool enables you to map one or more source elements to target elements. For example, you can map incoming source purchase order schema data to outgoing invoice schema data.

To create an XSL map file in Oracle BPEL Process Manager:

1. From the Component Palette, drag a **transform** activity into your BPEL process diagram. [Figure 45-4](#) provides an example.

Figure 45-4 Transform Activity



2. Double-click the **transform** activity.

The Transform dialog shown in [Figure 45-5](#) appears.

Figure 45-5 Transform Dialog

Variable	Part
inputVariable	payload

Target Variable:

Target Part:

Mapper File:

3. Specify the following information:

- a. Add source variables from which to map elements by clicking the **Add** icon and selecting the variable and part of the variable as needed (for example, a payload schema consisting of a purchase order request).

Note: You can select multiple input variables. The first variable defined represents the main XML input to the XSL map. Additional variables that are added here are defined in the XSL map as input parameters.

- b. Add target variables to which to map elements.
 - c. Add the target part of the variable (for example, a payload schema consisting of an invoice) to which to map.
4. In the **Mapper File** field, specify a map file name or accept the default name. The map file is the file in which you create your mappings using the XSLT Mapper transformation tool.
 5. Click the **Add** icon (second icon to the right of the Mapper File field) to create a mapping. If the file exists, click the **Edit** icon (third icon) to edit the mapping.
The XSLT Mapper appears.
 6. Go to [Section 45.1, "Introduction to the XSLT Mapper"](#) for an overview of using the XSLT Mapper.

45.2.2 How to Create an XSL Map File from Imported Source and Target Schema Files in Oracle BPEL Process Manager

Note: If you select a file with a `.xslt` extension such as `xform.xslt`, it opens the XSLT Mapper to create an XSL file named `xform.xslt.xsl`, even though your intention was to use the existing `xform.xslt` file. A `.xsl` extension is appended to *any* file that does not have a `.xsl` extension, and you must create the mappings in the new file. As a work around, ensure that your files first have an extension of `.xsl`. If the XSL file has an extension of `.xslt`, then rename it to `.xsl`.

The following steps provide a high level overview of how to create an XSL map in Oracle BPEL Process Manager using a `po.xsd` file and `invoice.xsd` file.

To create an XSL map file from imported source and target schema files in Oracle BPEL Process Manager:

1. In Oracle JDeveloper, select the application project in which you want to create the new XSL map.
2. Import the `po.xsd` and `invoice.xsd` files into the project (for example, in the Structure window of Oracle JDeveloper, right-click **Schemas** and select **Import Schemas**).
3. Right-click the selected project and select **New**.
The New Gallery dialog appears.
4. In the **Categories** tree, expand **General** and select **XML**.

5. In the **Items** list, double-click **XSL Map**.

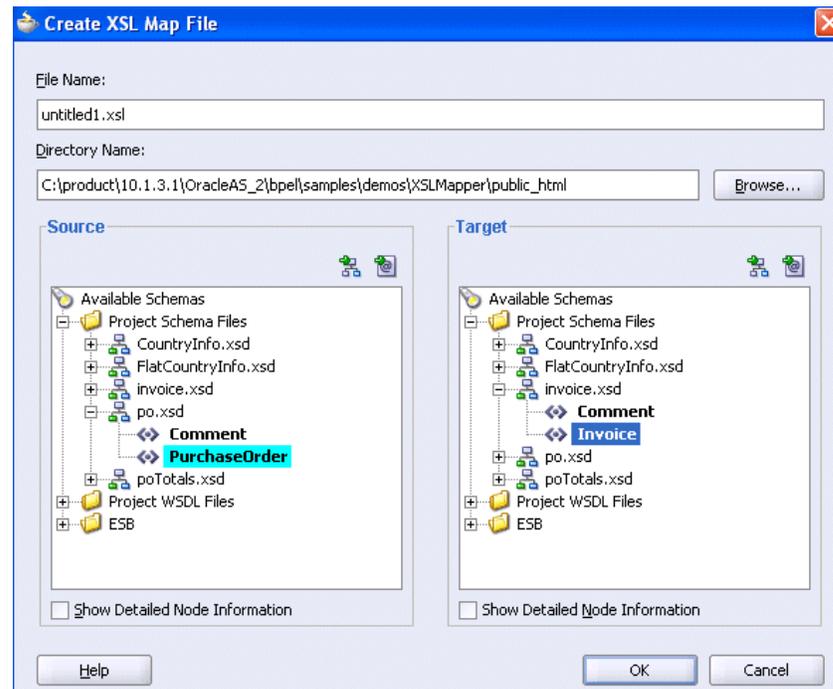
The Create XSL Map File dialog appears. This dialog enables you to create an XSL map file that maps a root element of a source schema file or Web Services Description Language (WSDL) file to a root element of a target schema file or WSDL file. Note the following details:

- WSDL files that have been added to the project appear under **Project WSDL Files**.
- Schema files that have been added to the project appear under **Project Schema Files**.
- Schema files that are not part of the project can be imported using the **Import Schema File** facility. Click the **Import Schema File** icon (first icon to the right and above the list of schema files).
- WSDL files that are not part of the project can be imported using the **Import WSDL File** facility. Click the **Import WSDL File** icon (second icon to the right and above the list of schema files).

6. Enter a name for the XSL map file in the **File Name** field.

7. Select the root element for the source and target trees. In the example in [Figure 45–6](#), the **PurchaseOrder** element is selected for the source root element and the **Invoice** element is selected for the target root element.

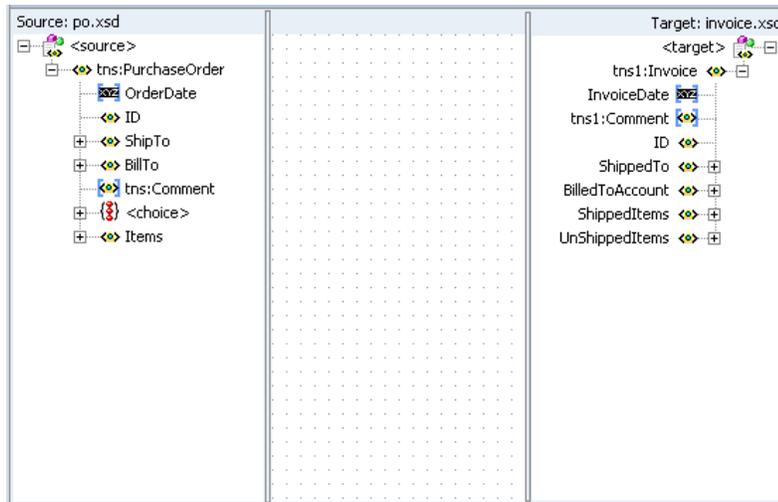
Figure 45–6 Expanded Target Section



8. Click **OK**.

A new XSL map is created, as shown in [Figure 45–7](#).

Figure 45–7 New XSL Map



9. Save and close the file now or begin to design your transformation. Information on using the XSLT Mapper is provided in [Section 45.1, "Introduction to the XSLT Mapper."](#)
10. From the Component Palette, drag a **transform** activity into your BPEL process.
11. Double-click the **transform** activity.
12. Specify the following information:
 - a. Add source variables from which to map elements by clicking the **Add** icon and selecting the variable and part of the variable as needed (for example, a payload schema consisting of a purchase order request).

Note: You can select multiple input variables. The first variable defined represents the main XML input to the XSL map. Additional variables that are added here are defined in the XSL map as input parameters.

- b. Add target variables to which to map elements.
 - c. Add the target part of the variable (for example, a payload schema consisting of an invoice) to which to map.
13. To the right of the **Mapper File** field, click the **Search** icon (first icon) to browse for the map file name you specified in Step 6.
14. Click **Open**.
15. Click **OK**.
The XSLT Mapper displays your XSL map file.
16. Go to [Section 45.1, "Introduction to the XSLT Mapper"](#) for an overview of using the XSLT Mapper.

45.2.3 How to Create an XSL Map File in Oracle Mediator

The XSLT Mapper enables you to create an XSL file to transform data from one XML schema to another in Oracle Mediator. After you define an XSL file, you can reuse it in

multiple routing rule specifications. This section provides an overview of creating a transformation map XSL file with the XSLT Mapper.

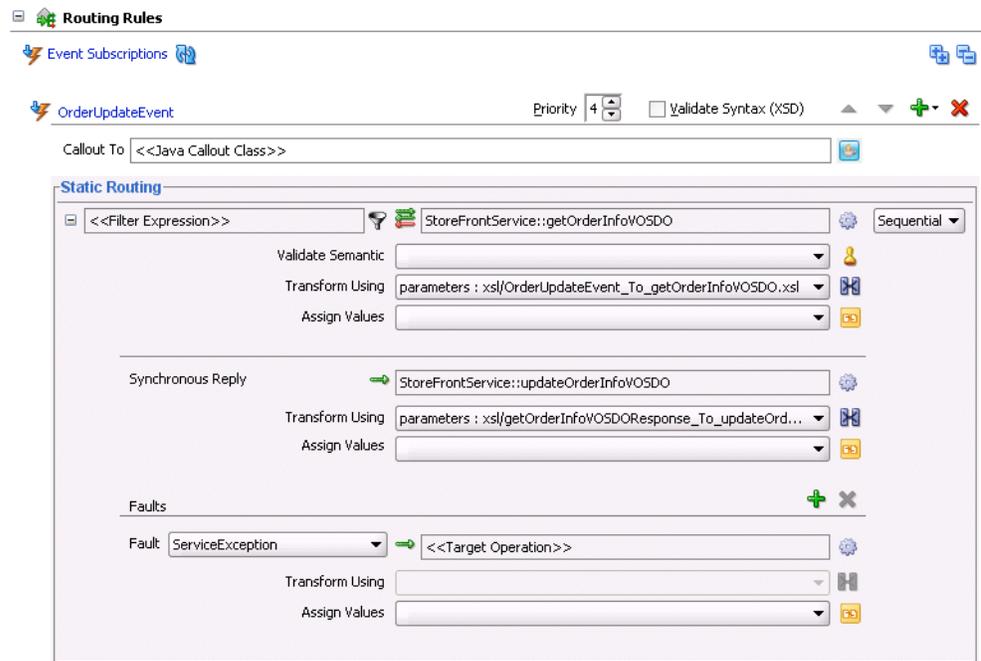
The XSLT Mapper is available from the Application Navigator in Oracle JDeveloper by clicking an XSL file or from the Oracle Mediator Editor by clicking the **transformation** icon, as described in the following steps. You can either create a new transformation map or update an existing one.

To launch the XSLT Mapper from the Mediator Editor and create or update a data transformation XSL file, follow these steps.

To create an XSL map file in Oracle Mediator:

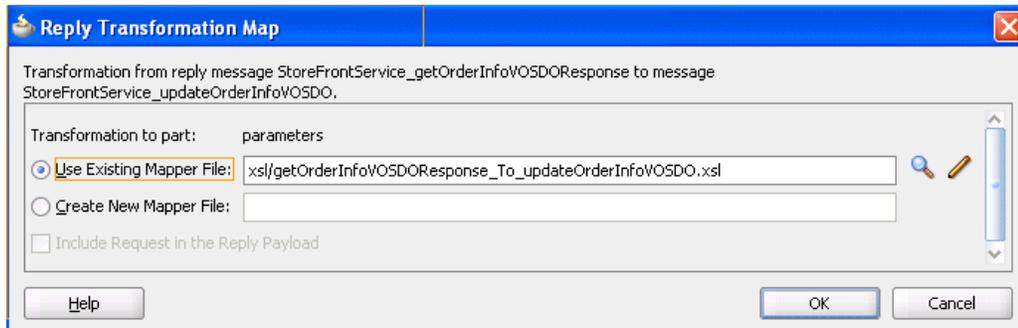
1. Open the Oracle Mediator Editor.
2. To the left of **Routing Rules**, click the + icon to open the **Routing Rules** panel. The **transformation map** icon is visible in the routing rules panel.
3. To the right of the **Transform Using** field shown in [Figure 45–8](#), click the appropriate **transformation map** icon to open the Transformation Map dialog.

Figure 45–8 Routing Rules



The appropriate Transformation Map dialog displays with options for selecting an existing transformation map (XSL) file or creating a new map file. For example, if you select the **transformation map** icon in the **Synchronous Reply** section, the dialog shown in [Figure 45–9](#) appears.

Figure 45–9 Reply Transformation Map Dialog

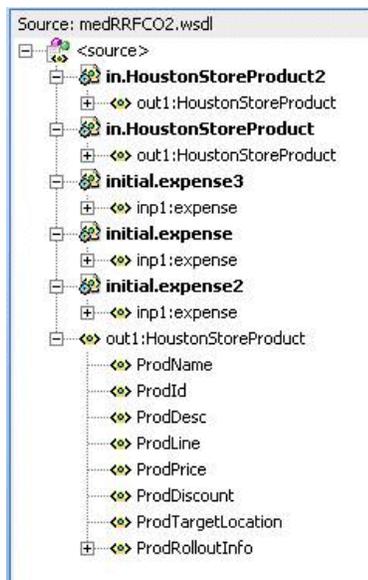


If the routing rule includes a synchronous reply or fault, the Reply Transformation Map dialog or Fault Transformation Map dialog contains the **Include Request in the Reply Payload** option. When you enable this option, you can obtain information from the request message. The request message and the reply and fault message can consist of multiple parts, meaning you can have multiple source schemas. Callback and callback timeout transformations can also consist of multiple parts.

Each message part includes a variable. For a reply transformation, the reply message includes a schema for the main part (the first part encountered) and an **in.partname** variable for each subsequent part. The include request message includes an **initial.partname** variable for each part.

For example, assume the main reply part is the **out1.HoustonStoreProduct** schema and the reply also includes two other parts that are handled as variables, **in.HoustonStoreProduct** and **in.HoustonStoreProduct2**. The request message includes three parts that are handled as the variables **initial.expense**, **initial.expense2**, and **initial.expense3**. Figure 45–10 provides an example.

Figure 45–10 Reply Part

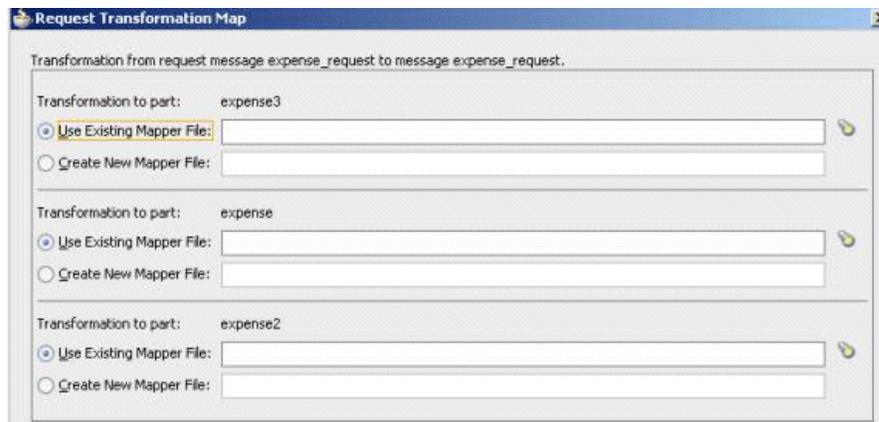


4. Choose one of the following options:
 - **Use Existing Mapper File** and then click the **Search** icon to browse for an existing XSLT Mapper file (or accept the default value).

- **Create New Mapper File** and then enter a name for the file (or accept the default value). If the source message in the WSDL file has multiple parts, variables are used for each part, as mentioned in Step 3. When the target of a transformation has multiple parts, multiple transformation files map to these targets. In this case, the mediator's transformation dialog has a separate panel for each target part. For example, here is a request in which the target has three parts:

Figure 45–11 provides an example.

Figure 45–11 Request Transformation Map Dialog



5. Click **OK**.

If you chose **Create New Mapper File**, the XSLT Mapper opens to enable you to correlate source schema elements to target schema elements.

6. Go to [Section 45.1, "Introduction to the XSLT Mapper"](#) for an overview of using the XSLT Mapper.

45.2.4 What You May Need to Know About Creating an XSL Map File

XSL file errors do not display during a transformation at runtime if you manually remove all existing mapping entries from an XSL file except for the basic format data. Ensure that you always specify mapping entries. For example, assume you perform the following actions:

1. Create a transformation mapping of input data to output data in the XSLT Mapper.
2. Design the application to write the output data to a file using the file adapter.
3. Manually modify the XSL file and remove all mapping entries except the basic format data. For example:

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet version="1.0"
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:ns0="http://xmlns.oracle.com/pcbpel/adapter/file/MediatorDemo/ValidationUsingSchematron/WriteAccountInfoToFile/"
xmlns:orcl="http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.ExtFunc"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:dvm="http://www.oracle.com/XSL/Transform/java/oracle.tip.dvm.LookupValue
```

```

"
xmlns:hwf="http://xmlns.oracle.com/bpel/workflow/xpath"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:mhdr="http://www.oracle.com/XSL/Transform/java/oracle.tip.mediator.service.common.functions.GetRequestHeaderExtnFunction"
xmlns:ids="http://xmlns.oracle.com/bpel/services/IdentityService/xpath"
xmlns:impl="http://www.mycompany.com/MyExample/NewAccount"
xmlns:tns="http://oracle.com/sca/soapservice/MediatorDemo/ValidationUsingSchematron/CreateNewCustomerService"
xmlns:xref="http://www.oracle.com/XSL/Transform/java/oracle.tip.xref.xpath.XRefXPathFunctions"
xmlns:plt="http://schemas.xmlsoap.org/ws/2003/05/partner-link/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:ora="http://schemas.oracle.com/xpath/extension"
xmlns:inp1="http://www.mycompany.com/MyExample/NewCustomer"
exclude-result-prefixes="xsi xsl tns xsd inp1 ns0 impl plt xp20 bpws orcl dvm
hwf mhdr ids xref ora">
</xsl:stylesheet>

```

While the file can still be compiled, the XSL mapping is now invalid.

4. Deploy and create an instance of the SOA composite application.

During instance creation, an exception error occurs when the write operation fails because it did not receive any input. However, note that no errors displayed during XSL transformation.

45.2.5 What Happens at Runtime If You Pass a Payload Through Oracle Mediator Without Creating an XSL Map File

If you design a SOA composite application to pass a payload through Oracle Mediator without defining any transformation mapping, Oracle Mediator passes the payload through. However, for the payload to be passed through successfully, you must ensure that your source and target message part names are the same and of the same type. Otherwise, the target reference may fail to execute with error messages such as Input source like Null or Part not found.

45.3 Designing Transformation Maps with the XSLT Mapper

The following sections describe how to use the XSLT Mapper in Oracle BPEL Process Manager or Oracle Mediator.

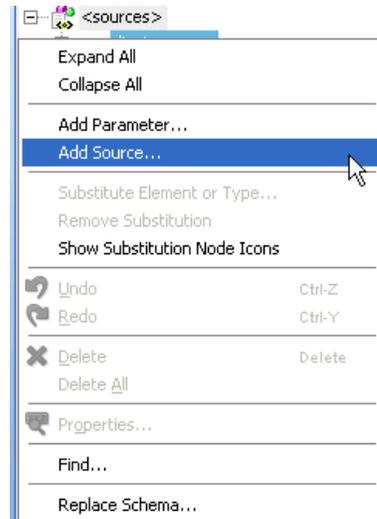
45.3.1 How to Add Additional Sources

You can add additional sources to an existing XSLT map. These sources are defined as global parameters and have schema files defining their structure. Multiple source documents may be required in certain instances depending upon the logic of the map. For instance, to produce an invoice, the map may need access to both a purchase order and a customer data document as input.

Note that XSL has no knowledge of BPEL variables. When you add multiple sources in XSL design time, ensure that you also add these multiple sources in the transform activity of a BPEL process.

To add additional sources:

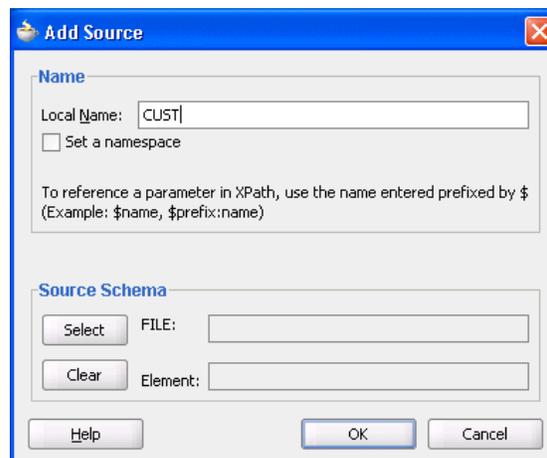
1. Right-click the source panel to display the context menu. [Figure 45–12](#) provides details.

Figure 45–12 Context Menu

2. Select **Add Source**.

The Add Source dialog shown in [Figure 45–13](#) appears.

3. Enter a parameter name for the source (the name can also be qualified by a namespace and prefix).

Figure 45–13 Add Source Dialog

4. In the **Source Schema** section, click **Select** to select a schema for the new source.

The Type Chooser dialog appears.

5. Select or import the appropriate schema or WSDL file for the parameter in the same manner as when creating a new XSLT map. For this example, the **Customer** element from the sample **customer.xsd** file is selected.

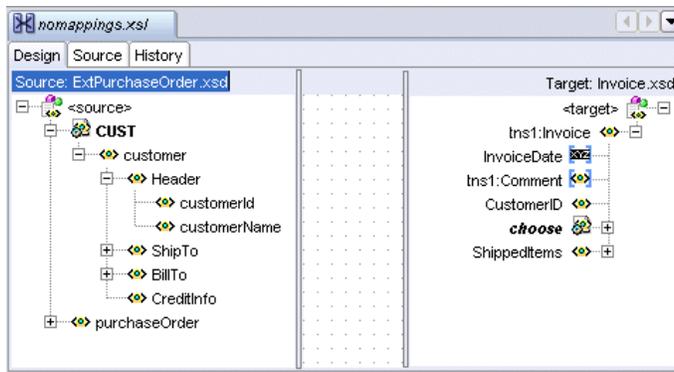
6. Click **OK**.

The schema definition appears in the **Source Schema** section of the Create Source as Parameter dialog.

7. Click **OK**.

The selected schema is imported and the parameter appears in the source panel above the main source. The parameter can be expanded as shown in [Figure 45–14](#) to view the structure of the underlying schema.

Figure 45–14 Expanded Parameter



The parameter can be referenced in XPath expressions by prefacing it with a \$. For example, a parameter named **CUST** appears as **\$CUST** in an XPath expression. Nodes under the parameter can also be referenced (for example, **\$CUST/customer/Header/customerid**).

45.3.2 How to Perform a Simple Copy by Linking Nodes

To copy an attribute or leaf-element in the source to an attribute or leaf-element in the target, drag the source to the target. For example, copy the element **PurchaseOrder/ID** to **Invoice/ID** and the attribute **PurchaseOrder/OrderDate** to **Invoice/InvoiceDate**, as shown in [Figure 45–15](#).

Figure 45–15 Linking Nodes



45.3.3 How to Set Constant Values

Perform the following steps to set a constant value.

To set constant values:

1. Select a node in the target tree.
2. Invoke the context menu by right-clicking the mouse.
3. Select the **Set Text** menu option.

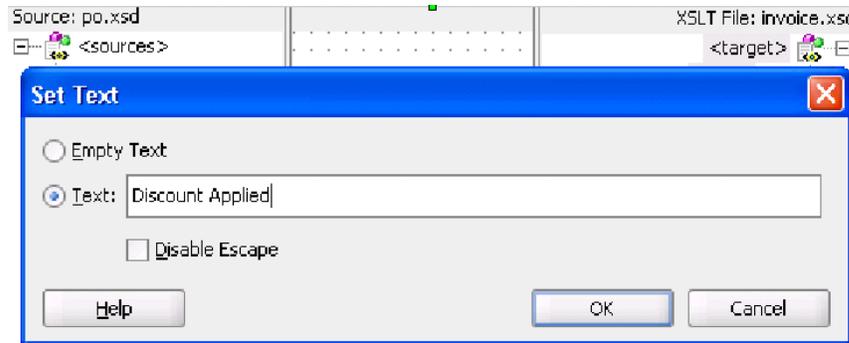
A menu provides the following selections:

- **<Empty>**: Enables you to create an empty node.
 - **Enter Text**: Enables you to enter text.
4. Select **Enter Text**.

The Set Text dialog appears.

5. In the Set Text dialog, enter text (for example, **Discount Applied**, as shown in Figure 45–16).

Figure 45–16 Set Text Dialog



6. Click **OK** to save the text.
A **T** icon is displayed next to the node that has text associated with it. The beginning of the text that is entered is shown next to the node name.
7. If you want to modify the text associated with the node, right-click the node and select **Edit Text** to invoke the Set Text dialog again.
8. Edit the contents and click **OK**.
For more information about the fields, see the online Help for the Set Text dialog.
9. If you want to remove the text associated with the node, right-click the node and select **Remove Text**.

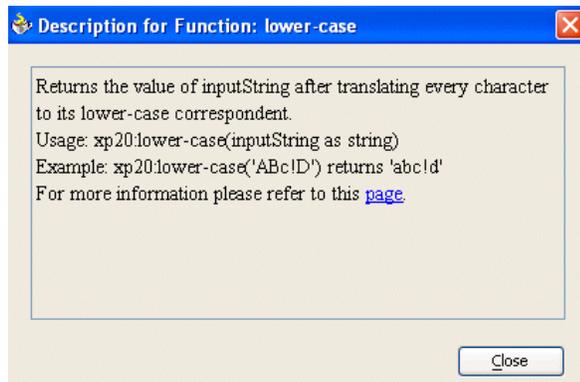
45.3.4 How to Add Functions

In addition to the standard XPath 1.0 functions, the XSLT Mapper provides many prebuilt extension functions and can support user-defined functions and named templates. The extension functions are prefixed with **oraext** or **orcl** and mimic XPath 2.0 functions.

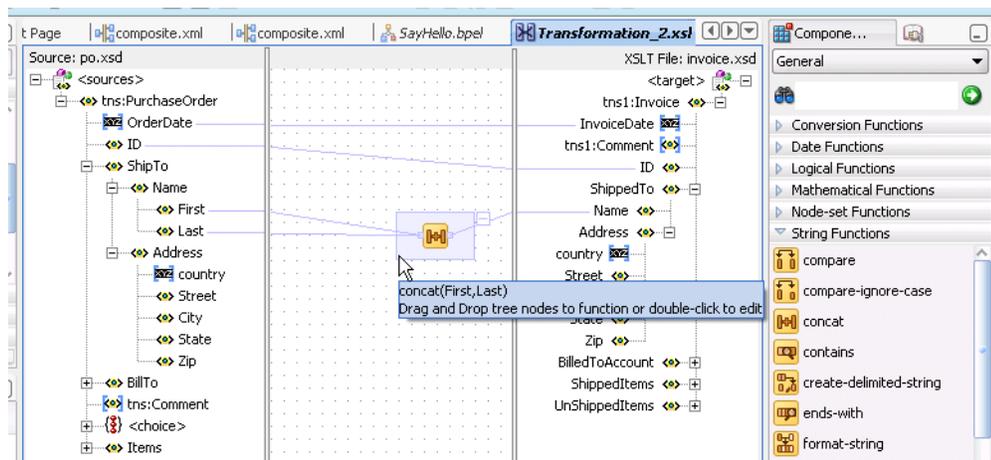
Perform the following steps to view function definitions and use a function.

To add functions:

1. From the Component Palette, select a category of functions (for example, **String Functions**).
2. Right-click an individual function (for example, **lower-case**).
3. Select **Help**. A dialog with a description of the function appears, as shown in Figure 45–17. You can also click a link at the bottom to access this function's description at the World Wide Web Consortium at www.w3.org.

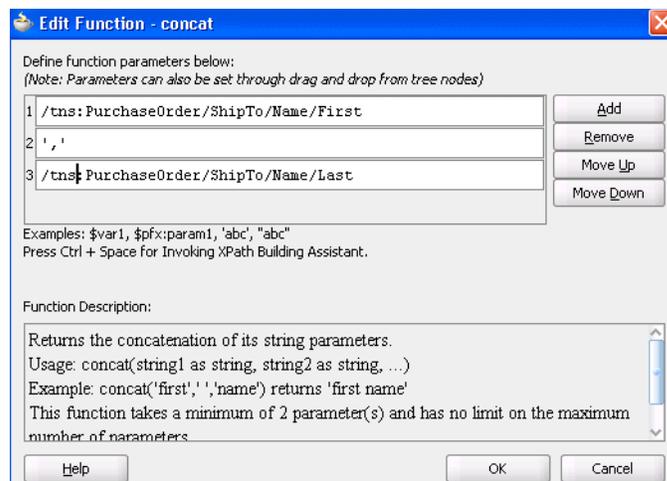
Figure 45–17 Description of Function

4. Drag a function from the Component Palette to the center panel of the XSLT Mapper. You can then connect the source parameters from the source tree to the function and the output of the function to a node in the target tree. For the following example, drag the **concat** function from the **String** section of the Component Palette to the center panel.
5. Concatenate **PurchaseOrder/ShipTo/Name/First** and **PurchaseOrder/ShipTo/Name/Last**. Place the result in **Invoice/ShippedTo/Name** by dragging threads from the first and last names and dropping them on the input (left) side on the **concat** function.
6. Drag a thread from the **ShippedTo** name and connect it to the output (right) side on the **concat** function, as shown in [Figure 45–18](#).

Figure 45–18 Using the Concat Function

45.3.4.1 Editing Function Parameters

To edit the parameters of any function, double-click the function icon to launch the Edit Function dialog. For example, if you want to add a new comma parameter so that the output of the **concat** function used in the previous example is **Last, First**, then click **Add** to add a comma and reorder the parameters to get this output. [Figure 45–19](#) provides details.

Figure 45–19 Editing Function Parameters

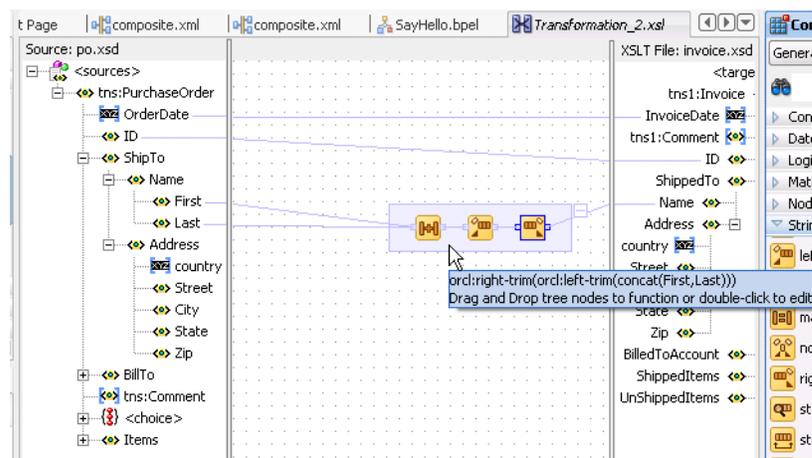
For more information about how to add, remove, and reorder function parameters, see the online Help for the Edit Function dialog.

45.3.4.2 Chaining Functions

Complex expressions can be built by chaining functions (that is, mapping the output of one function to the input of another). For example, to remove all leading and trailing spaces from the output of the **concat** function, perform the following steps:

1. Drag the left-trim and right-trim functions into the border area of the **concat** function.
2. Chain them as shown in [Figure 45–20](#) by dragging lines from the output side of one function to the input side of the next function.

Chaining can also be performed by dragging and dropping a function onto a connecting link.

Figure 45–20 Chaining Functions

45.3.4.3 Using Named Templates

Some complicated mapping logic cannot be represented or achieved by visual mappings. For these situations, named templates are useful. Named templates enable

you to share common mapping logic. You can define the common mapping logic as a named template and then use it as often as you want.

You can define named templates in two ways:

- Add the template directly to your XSL map in source view.
- Add the template to an external file that you include in your XSL map.

The templates you define appear in the **User Defined Named Templates** list of the **User Defined** page in the Component Palette. You can use named templates in almost the same way as you use other functions. The only difference is that you cannot link the output of a named template to a function or another named template; you can only link its output to a target node in the target tree.

To create named templates, you must be familiar with the XSLT language. See any XSLT book or visit the following URL for details about writing named templates:

<http://www.w3.org/TR/xslt>

For more information about including templates defined in external files, see [Section 45.3.6.7, "Including External Templates with `xsl:include`."](#)

45.3.4.4 Importing User-Defined Functions

You can create and import a user-defined Java function if you have complex functionality that cannot be performed in XSLT or with XPath expressions.

Follow these steps to create and use your own functions. External, user-defined functions can be necessary when logic is too complex to perform within the XSL map.

To import user-defined functions:

1. Code and build your functions.

The XSLT Mapper extension functions are coded differently than the Oracle BPEL Process Manager extension functions. Two examples are provided in the `SampleExtensionFunctions.java` file of the `mapper-107-extension-functions` sample scenario. [Example 45-5](#) provides the text for these functions. To download these and other samples, visit the following URL:

http://www.oracle.com/technology/sample_code/products/soa

Each function must be declared as a static function. Input parameters and the returned value must be declared as one of the following types:

- `java.lang.String`
- `int`
- `float`
- `double`
- `boolean`
- `oracle.xml.parser.v2.XMLNodeList`
- `oracle.xml.parser.v2.XMLDocumentFragment`

Example 45-5 XSLT Mapper Extension Functions

```
// SampleExtensionFunctions.java
package oracle.sample;
```

```

/*
This is a sample XSLT Mapper User Defined Extension Functions implementation
class.
*/
public class SampleExtensionFunctions
{
    public static Double toKilograms(Double lb)
    {
        return new Double(lb.doubleValue()*0.45359237);
    }
    public static String replaceChar(String inputString, String oldChar, String
        newChar )
    {
        return inputString.replace(oldChar.charAt(0), newChar.charAt(0));
    }
}

```

2. Create an XML extension function configuration file. This file defines the functions and their parameters.

This file must have the name `ext-mapper-xpath-functions-config.xml`. See [Section B.7, "Creating User-Defined XPath Extension Functions"](#) for more information on the format of this file. The file shown in [Example 45–6](#) represents the functions `toKilograms` and `replaceChar` as they are coded in [Example 45–5](#).

Example 45–6 XML Extension Function Configuration File

```

<?xml version="1.0" encoding="UTF-8"?>
<soa-xpath-functions version="11.1.1"
  xmlns="http://xmlns.oracle.com/soa/config/xpath" xmlns:sample=
"http://www.oracle.com/XSL/Transform/java/oracle.sample.SampleExtensionFunctions"
  >
    <function name="sample:toKilograms">
      <className>oracle.sample.SampleExtensionFunctions</className>
      <return type="number"/>
      <params>
        <param name="pounds" type="number"/>
      </params>
      <desc>Converts a value in pounds to kilograms</desc>
    </function>
    <function name="sample:replaceChar">
      <className>oracle.sample.SampleExtensionFunctions</className>
      <return type="string"/>
      <params>
        <param name="inputString" type="string"/>
        <param name="oldChar" type="string"/>
        <param name="newChar" type="string"/>
      </params>
      <desc>Returns a new string resulting from replacing all occurrences
        of oldChar in this string with newChar</desc>
    </function>
  </soa-xpath-functions>

```

Some additional rules apply to the definitions of XSLT extension functions:

- The functions need a namespace prefix and a namespace. In this sample, they are `sample` and `http://www.oracle.com/XSL/Transform/java/oracle.sample.SampleExtensionFunctions`.

- The function namespace must start with `http://www.oracle.com/XSL/Transform/java/` for extension functions to work with the Oracle XSLT processor.
- The last portion of the namespace, in this sample `oracle.sample.SampleExtensionFunctions`, must be the fully qualified name of the Java class that implements the extension functions.
- The types and their equivalent Java types shown in [Table 45–1](#) can be used for parameter and return values:

Table 45–1 Types and Equivalent Java Types

XML Configuration File Type Name	Java Type
string	<code>java.lang.String</code>
boolean	<code>boolean</code>
number	<code>int, float, double</code>
node-set	<code>oracle.xml.parser.v2.XMLNodeList</code>
tree	<code>oracle.xml.parser.v2.XMLDocumentFragment</code>

3. Create a JAR file containing both the XML configuration file and the compiled classes. The configuration file must be contained in the `META-INF` directory for the JAR file. For the example in this section, the directory structure is as follows with the `oracle` and `META-INF` directories added to a JAR file:
 - `oracle`
 - `sample` (contains the class file)
 - `META-INF`
 - `ext-mapper-xpath-functions-config.xml`

The JAR file must then be registered with Oracle JDeveloper.

4. Go to **Tools > Preferences > SOA**.
5. Click the **Add** button and navigate to and select your JAR file.
6. Restart Oracle JDeveloper.

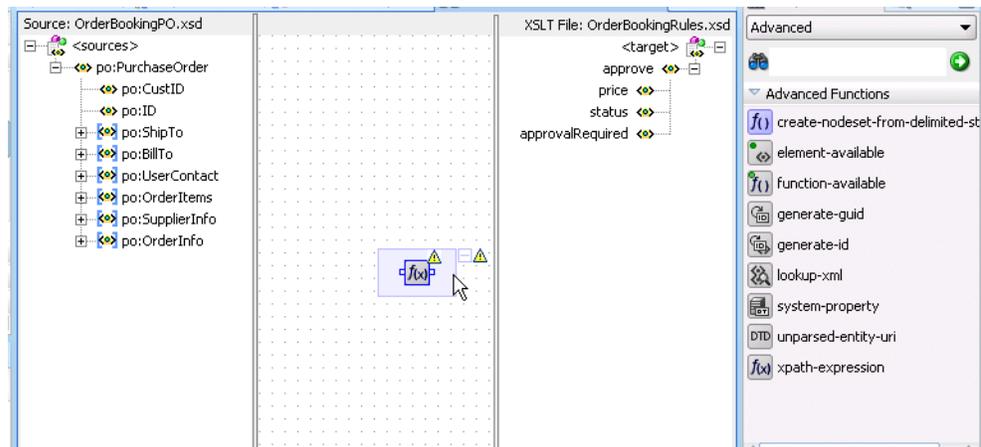
New functions appear in the Component Palette under the **User Defined** page in the **User Defined Extension Functions** group.

7. To make the functions available in the runtime environment, [Section B.7.3, "How to Deploy User-Defined Functions to Runtime"](#) for details.

45.3.5 How to Edit XPath Expressions

To use an XPath expression in a transformation mapping, select the **Advanced** page and then the **Advanced Function** group from the Component Palette and drag `xpath-expression` from the list into the XSLT Mapper. This is shown in [Figure 45–21](#).

Figure 45–21 *Editing XPath Expressions*



When you double-click the icon, the Edit XPath Expression dialog appears, as shown in Figure 45–22. You can press Ctrl+Space to invoke the XPath Building Assistant.

Figure 45–22 *Edit XPath Expression Dialog*

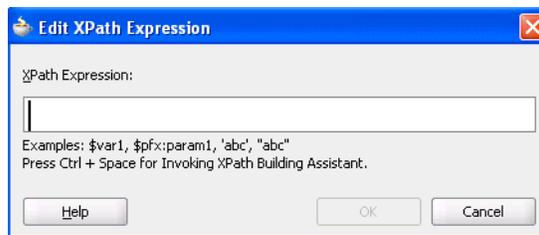
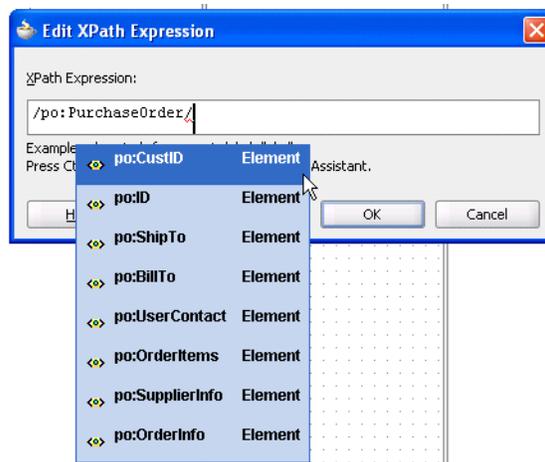


Figure 45–23 shows the XPath Building Assistant.

Figure 45–23 *The XPath Building Assistant*



For more information about using the XPath Building Assistant, see the online Help for the Edit XPath Expression dialog.

45.3.6 How to Add XSLT Constructs

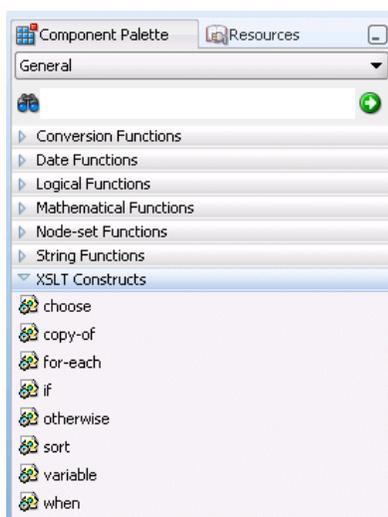
While mapping complex schemas, it is essential to be able to add XSLT constructs. For instance, you may need to create a node in the target when a particular condition exists; this requires the use of an `xsl:if` statement or an `xsl:choose` statement. You may also need to loop over a `node-set` in the source such as a list of items in a sales order and create nodes in the target XML for each item in the sales order; this requires the use of an `xsl:for-each` statement. The XSLT Mapper provides XSLT constructs for performing these and other tasks.

There are two ways to add XSLT constructs such as **for-each**, **if**, or **choose** to the target XSLT tree:

To add XSLT constructs from the Component Palette:

1. Select the **General** page and open the **XSLT Constructs** group. [Figure 45–24](#) provides details.

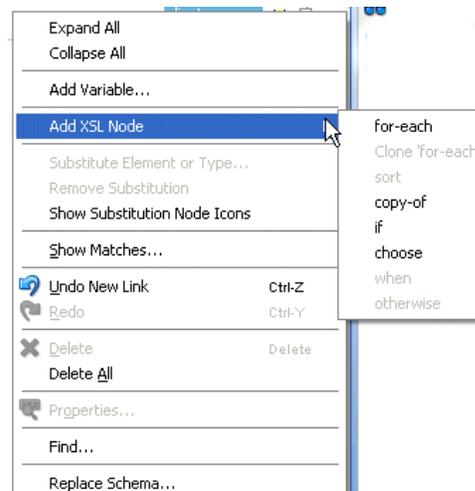
Figure 45–24 XSLT Constructs Available Through the Component Palette



2. Drag an XSLT construct from the group onto a node in the target tree. If the XSLT construct can be applied to the node, it is inserted in the target tree. Note that the **when** and **otherwise** constructs must be applied to a previously-inserted **choose** node.

To add XSLT constructs through the context menu on the target tree:

1. Right-click the element in the target tree where you want to insert an XSLT construct. A context menu is displayed. [Figure 45–25](#) provides details.

Figure 45–25 XSLT Constructs in Available Through the Context Menu

2. Select **Add XSL Node** and then the XSLT construct you want to insert.

The XSLT construct is inserted. In most cases, an error icon initially appears next to the construct. This indicates that the construct requires an XPath expression to be defined for it.

In the case of the **for-each** construct, for example, an XPath expression defines the node set over which the **for-each** statement loops. In the case of the **if** construct, the XPath expression defines a boolean expression that is evaluated to determine if the contents of the **if** construct are executed.

The XPath expression can be created in the same manner as mapping elements and attributes in the target tree. The following methods create an underlying XPath expression in the XSLT. You can perform all of these methods on XSLT constructs in the target tree to set their XPath expressions:

- Creating a simple copy by linking nodes
- Adding functions
- Adding XPath expressions

The following sections describe specific steps for inserting each supported XSLT construct.

45.3.6.1 Using Conditional Processing with `xsl:if`

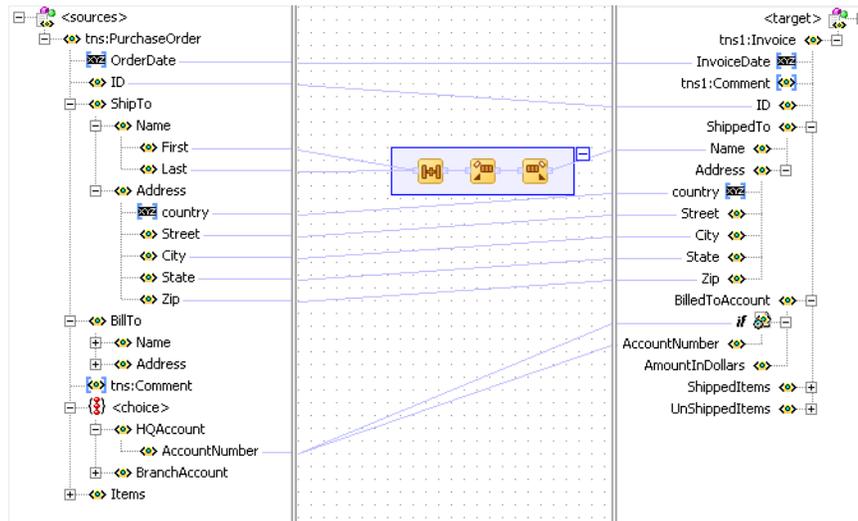
In [Figure 45–26](#), note that **HQAccount** and **BranchAccount** are part of a choice in the **PurchaseOrder** schema; only one of them exists in an actual instance. To illustrate conditional mapping, copy **PurchaseOrder/HQAccount/AccountNumber** to **Invoice/BilledToAccount/AccountNumber**, only if it exists.

To use conditional processing with `xsl:if`:

1. In the target tree, select **Invoice/BilledToAccount/AccountNumber** and right-click to invoke the context sensitive menu.
2. Select **Add XSL Node > if** and connect **PurchaseOrder/HQAccount/AccountNumber** to **Invoice/BilledToAccount/if**.
3. Connect **PurchaseOrder/HQAccount/AccountNumber** to **Invoice/BilledToAccount/if/AccountNumber**.

Figure 45–26 shows the results.

Figure 45–26 Conditional Processing with `xsl:if`



When mapping an optional source node to an optional target node, it is important to surround the mapping with an `xsl:if` statement that tests for the existence of the source node. If this is not done and the source node does not exist in the input document, an empty node is created in the target document. For example, note the mapping shown in [Example 45–7](#):

Example 45–7 Statement Without `xsl:if`

```
<ProductName>
  <xsl:value-of select="ProductName" />
</ProductName>
```

If the `ProductName` field is optional in both the source and target and the element does not exist in the source document, then an empty `ProductName` element is created in the target document. To avoid this situation, add an `if` statement to test for the existence of the source node before the target node is created, as shown in [Example 45–8](#):

Example 45–8 Statement With `xsl:if`

```
<xsl:if test="ProductName">
  <ProductName>
    <xsl:value-of select="ProductName" />
  </ProductName>
</xsl:if>
```

45.3.6.2 Using Conditional Processing with `xsl:choose`

In this same example, you can copy `PurchaseOrder/HQAccount/AccountNumber` to `Invoice/BilledToAccount/AccountNumber`, if it exists. Otherwise, copy `PurchaseOrder/BranchAccount` to `Invoice/BilledToAccount/AccountNumber`.

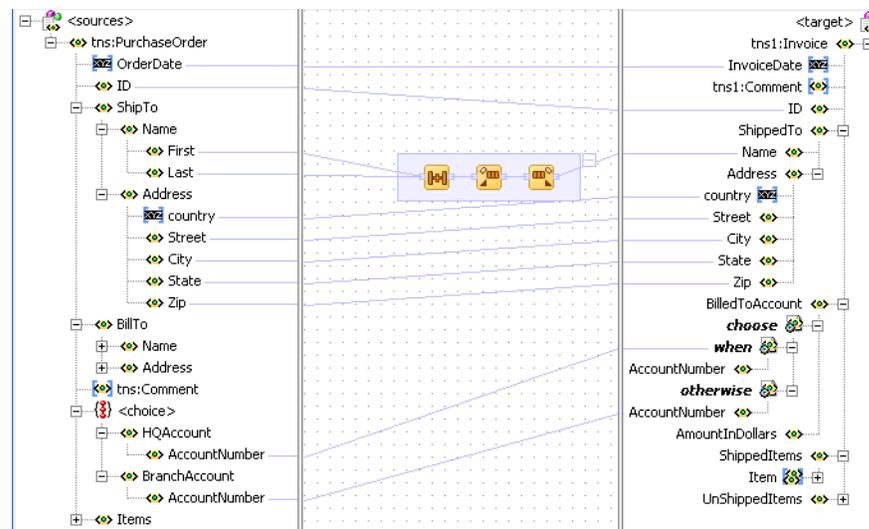
To use conditional processing with `xsl:choose`:

1. In the target tree, select `Invoice/BilledToAccount/AccountNumber` and right-click to invoke the context sensitive menu.

2. Select **Add XSL Node** > **choose** from the menu.
3. Connect **PurchaseOrder/HQAccount/AccountNumber** to **Invoice/BilledToAccount/choose/when** to define the condition.
4. Connect **PurchaseOrder/HQAccount/AccountNumber** to **Invoice/BilledToAccount/choose/when/AccountNumber**.
5. In the target tree, select **XSL Add Node** > **choose** and right-click to invoke the context sensitive menu.
6. Select **Add XSL node** > **otherwise** from the menu.
7. Connect **PurchaseOrder/BranchAccount/AccountNumber** to **Invoice/BilledToAccount/choose/otherwise/AccountNumber**.

Figure 45–27 shows the results.

Figure 45–27 Conditional Processing with *xsl:choose*



45.3.6.3 Creating Loops with *xsl:for-each*

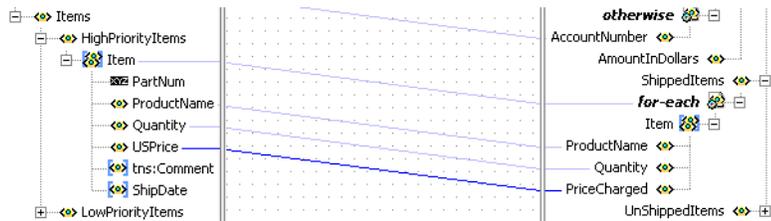
The XSLT Mapper enables you to create loops with the *xsl:for-each* command. For example, copy **PurchaseOrder/Items/HighPriorityItems/Item** to **Invoice/ShippedItems/Item**.

To create loops with *xsl:for-each*:

1. In the target tree, select **Invoice/ShippedItems/Item** and right-click to invoke the context sensitive menu.
2. Select **Add XSL Node** > **for-each** and connect **PurchaseOrder/Items/HighPriorityItems/Item** to **Invoice/ShippedItems/for-each** to define the iteration.
3. Connect **PurchaseOrder/Items/HighPriorityItems/Item/ProductName** to **Invoice/ShippedItems/for-each/Item/ProductName**.
4. Connect **PurchaseOrder/Items/HighPriorityItems/Item/Quantity** to **Invoice/ShippedItems/for-each/Item/Quantity**.
5. Connect **PurchaseOrder/Items/HighPriorityItems/Item/USPrice** to **Invoice/ShippedItems/for-each/Item/PriceCharged**.

Figure 45–28 shows the results.

Figure 45–28 Creating Loops with `xsl:for-each`



Notes:

- Executing an auto map automatically inserts `xsl:for-each`. To see the auto map in use, drag **PurchaseOrder/Items/LowPriorityItems** to **Invoice/UnShippedItems**; `for-each` is automatically created.
- Ensure that your design does not include infinite loops. These loops result in errors similar to the following displaying during deployment and invocation of your application.

```
ORAMED-04001:
.
.
.
oracle.tip.mediator.service.BaseActionHandler requestProcess
SEVERE:
failed reference BPELProcess1.bpelprocess1_client operation =
process
```

45.3.6.4 Cloning `xsl:for-each`

You can create additional loops by cloning an existing `xsl:for-each`. For example, copy all **LowPriorityItems** to **ShippedItems**, in addition to **HighPriorityItems**.

To clone `xsl:for-each`:

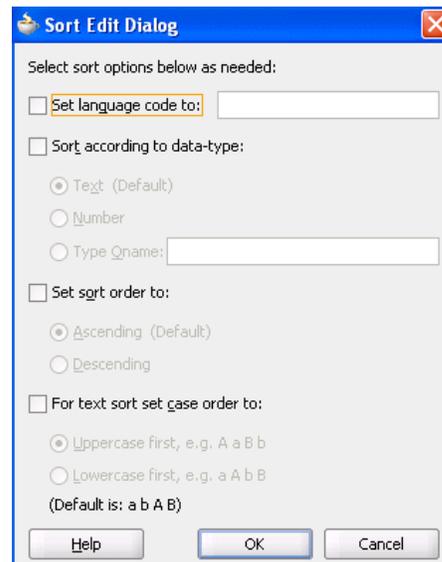
1. Under **Invoice/ShippedItems**, select `for-each`.
2. Right-click and select **Add XSL Node > Clone 'for-each'**.
This inserts a copy of the `for-each` node beneath the original `for-each`.
3. Drag **PurchaseOrder/Items/LowPriorityItems/Item** to the copied `for-each` to define the iteration.
4. Connect **PurchaseOrder/Items/LowPriorityItems/Item/ProductName** to **Item/ProductName** in the copied `for-each`.
5. Connect **PurchaseOrder/Items/LowPriorityItems/Item/Quantity** to **Item/Quantity** in the copied `for-each`.
6. Connect **PurchaseOrder/Items/LowPriorityItems/Item/USPrice** to **Item/PriceCharged** in the copied `for-each`.

45.3.6.5 Applying `xsl:sort` to `xsl:for-each`

The XSLT Mapper enables you to add `xsl:sort` statements to `xsl:for-each` commands.

To add an `xsl:sort` statement:

1. Right-click a **for-each** statement in the target tree.
A context menu appears.
2. Select **Add XSL Node > sort**. The Sort Edit Dialog is displayed.

Figure 45–29 Sort Edit Dialog

3. Select options to add to the sort statement as needed. See the online Help for information on options.
4. Click **OK**. The sort statement is added following the **for-each**.
5. To set the field on which to sort, drag from the necessary sort field node in the source tree to the sort node in the target tree. This creates a simple link and sets the XPath expression for the select attribute on the `xsl:sort`.
6. To add additional sort statements, right-click the **for-each** to add another sort or right-click an existing sort node to insert a new sort statement before the selected sort node.
7. To edit a sort node, double-click the sort node or right-click and select **Edit Sort** from the context menu. This invokes the Sort Edit Dialog and enables you to change the sort options.

45.3.6.6 Copying Nodes with `xsl:copy-of`

You may need to use the XSLT **copy-of** construct to copy a node, along with any child nodes, from the source to the target tree. This is typically done when working with **anyType** or **any** element nodes. Note that **anyType** and **any** element and attribute nodes cannot be mapped directly. Use **copy-of** or element and type substitution.

To copy nodes with `xsl:copy-of`:

1. Select the node in the target tree to be created by the **copy-of** command.
2. Right-click the node and select **Add XSL Node > copy-of**.

If the node is not an **any** element node, a dialog appears requesting you to either replace the selected node or replace the children of the selected node.

3. Select the correct option for your application and click **OK**.

If you select **Replace the selected node** with the **copy-of**, a processing directive is created immediately following the **copy-of** in the XSL indicating which node is replaced by the **copy-of**. Without the processing directive in the XSL, the conversion back to design view is interpreted incorrectly. For this reason, do not remove or edit this processing instruction while in source view.

4. Set the source node for the **copy-of** by dragging and dropping from the source tree or by creating an XPath expression.

Note: Always create the **copy-of** command in design view so that the correct processing directive can be created in the XSLT Mapper to indicate the correct placement of the **copy-of** command in the target tree.

WARNING: The XSLT Mapper does not currently validate the mapping of data performed through use of the **copy-of** command. You must ensure that **copy-of** is used to correctly map elements to the target tree so that the target XML document contains valid data. You can test the validity by using the test tool.

45.3.6.7 Including External Templates with `xsl:include`

You can reuse templates that are defined in external XSL files by including them in the current map with an include statement.

To include external templates with `xsl:include`:

1. In the target tree, select and right-click the root node.
2. From the menu, select **Add Include File**.

A dialog prompts you for the include file name.

3. Select the file and click **OK**.

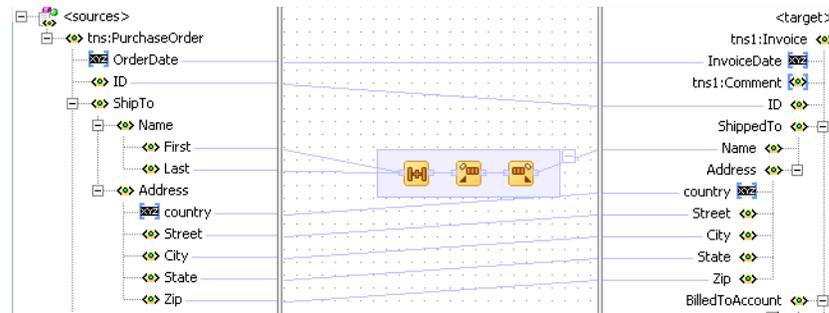
The file is copied to the same project directory as the existing map file. A relative path name is created for it and the include statement instruction is inserted in the target tree.

The include file can only contain named template definitions. These are parsed and available to you in design view of the Component Palette under the **User Defined Named Templates** category in the **User Defined** page.

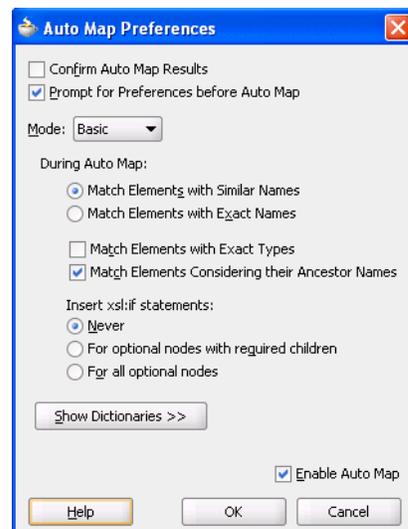
Note: An `orams://` shared location cannot be referenced for a user-defined named template include file.

45.3.7 How to Automatically Map Nodes

Mapping nonleaf nodes starts the auto map feature. The system automatically tries to link all relevant nodes under the selected source and target. Try the auto map feature by mapping **PurchaseOrder/ShipTo/Address** to **Invoice/ShippedTo/Address**. All nodes under **Address** are automatically mapped, as shown in [Figure 45-30](#).

Figure 45–30 Auto Mapping

The behavior of the auto map can be tuned by altering the settings in Oracle JDeveloper preferences or by right-clicking the XSLT Mapper and selecting **Auto Map Preferences**. This displays the dialog shown in Figure 45–31.

Figure 45–31 Auto Map Preferences

This dialog enables you to customize your auto mapping as follows:

- Invoke the automatic mapping feature, which attempts to automatically link all relevant nodes under the selected source and target. When disabled, you must individually map relevant nodes.
- Display and review all potential source-to-target mappings detected by the XSLT Mapper, and then confirm to create them.
- Be prompted to customize the auto map preferences before the auto map is invoked.
- Select the **Basic** or **Advanced** method for automatically mapping source and target nodes. This action enables you to customize how the XSLT Mapper attempts to automatically link all relevant nodes under the selected source and target.
- Manage your dictionaries. The XSLT Mapper uses the rules defined in a dictionary when attempting to automatically map source and target elements.

For more information on the fields, see the online Help for the Auto Map Preferences dialog.

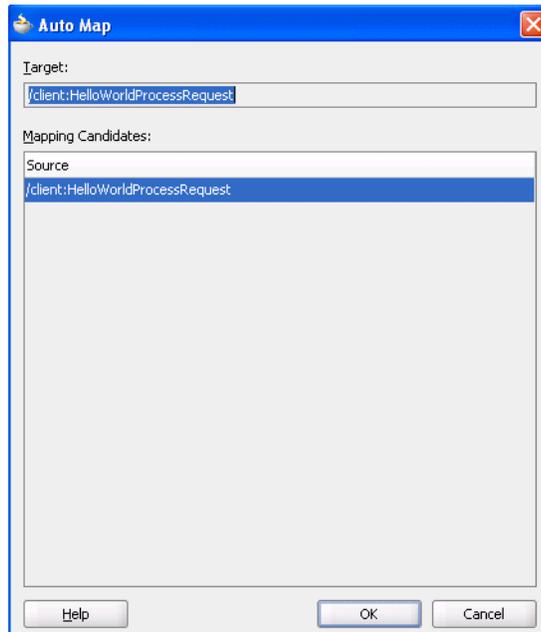
Follow these instructions to see potential source mapping candidates for a target node.

To automatically map nodes:

1. Right-click the target node and select **Show Matches**.
2. Click **OK** in the Auto Map Preferences dialog.

The Auto Map dialog appears, as shown in [Figure 45–32](#).

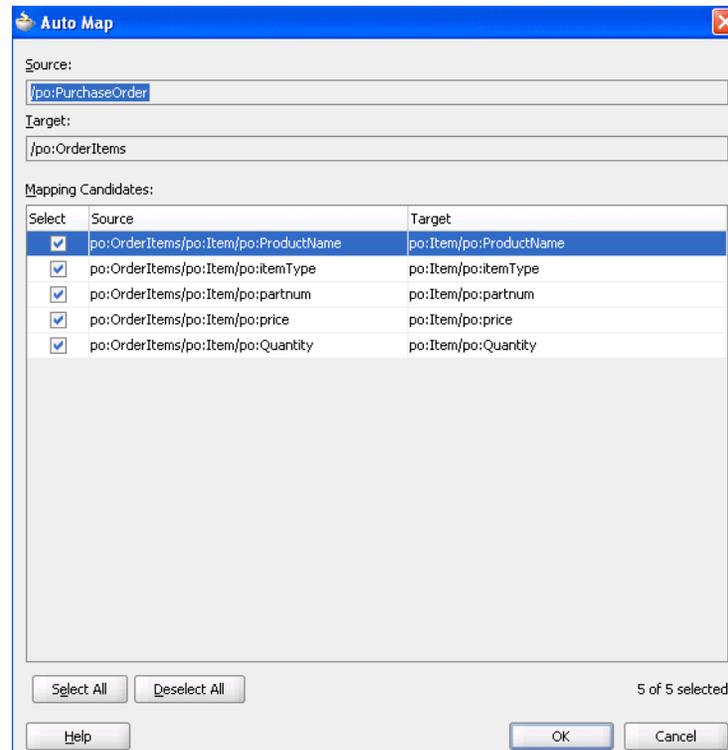
Figure 45–32 Auto Mapping Candidates



For more information on the fields, see the online Help for the Auto Map dialog.

45.3.7.1 Using Auto Mapping with Confirmation

When the **Confirm Auto Map Results** checkbox shown in [Figure 45–31](#) is selected, a confirmation dialog appears. If matches are found, the potential source-to-target mappings detected by the XSLT Mapper are displayed, as shown in [Figure 45–33](#). The dialog enables you to filter one or more mappings.

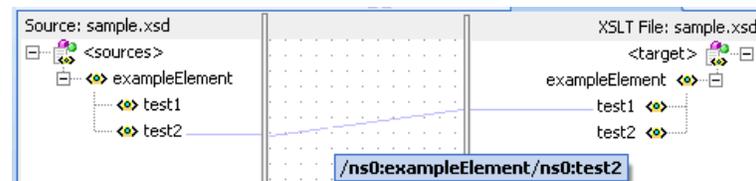
Figure 45–33 Auto Map with Confirmation

For more information about the fields, see the online Help for the Auto Map dialog.

45.3.8 What You May Need to Know About Automatic Mapping

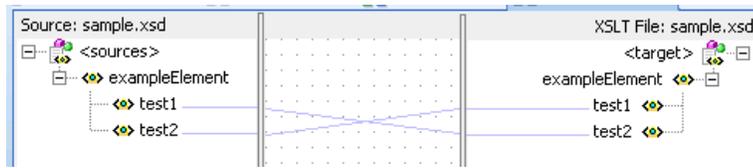
The automatic mapping algorithm depends on existing maps between source and target nodes. When maps exist between source and target nodes before executing automatic mapping, these existing maps are used to define valid synonyms that are used by the algorithm.

For example, assume you have a simple source and target tree, each with two elements called **test1** and **test2**, as shown in [Figure 45–34](#).

Figure 45–34 Source and Target Tree with Two Elements

If no nodes are mapped, the automatic mapping algorithm does not match the names **test1** and **test2**. However, if mapping exists between the **test1** and **test2** nodes, the algorithm predefines the names **test1** and **test2** as synonyms for any additional mapping.

In the example in [Figure 45–34](#), if you drag the **exampleElement** from the source to the target, the automatic mapping algorithm maps the **test1** node in the source to the **test2** node in the target because your map previously linked those two names. This results in the map shown in [Figure 45–35](#):

Figure 45–35 Results of Dragging exampleElement

45.3.9 How to View Unmapped Target Nodes

You can view a list of target nodes that are currently unmapped to source nodes.

To view unmapped target nodes:

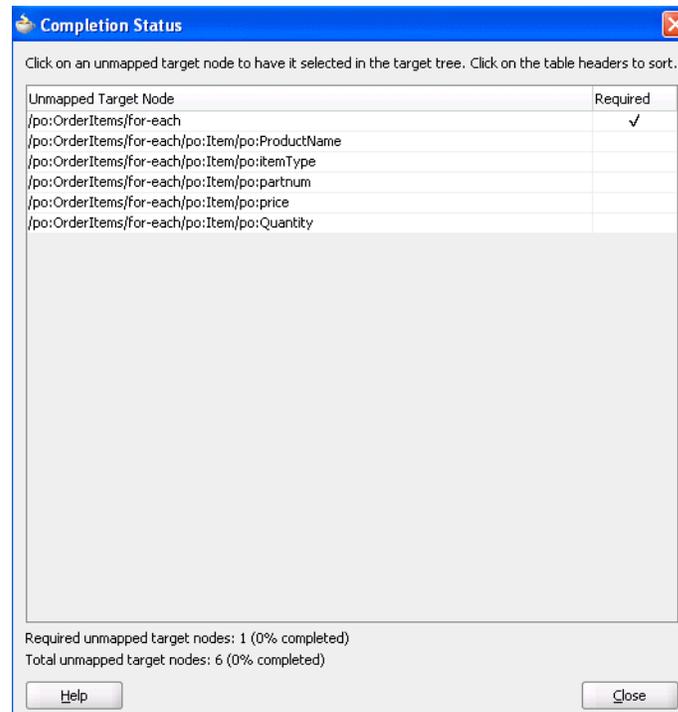
1. In the XSLT Mapper, right-click in the center panel and select **Completion Status**.

This dialog provides statistics at the bottom about the number of unmapped target nodes. This dialog enables you to identify and correct any unmapped nodes before you test your transformation mapping logic on the Test XSL Map dialog.

2. In the list, select a target node. The node is highlighted. A checkmark indicates that the target node is required to be mapped. If not required, the checkbox is empty.

Note: Nodes are marked as required in the Completion Status dialog based on the XSD definition for a node. It is possible that a node marked as required is not actually required for a specific mapping if a parent node of the required node is optional and is not part of the XSL mapping.

Figure 45–36 provides an example of the Completion Status dialog.

Figure 45–36 Completion Status

45.3.10 How to Generate Dictionaries

A dictionary is an XML file used by automatic mapping. It contains synonyms for field names. For instance, assume that the element **QtyOrdered** should map to the element **Quantity**. The element names **QtyOrdered** and **Quantity** are then synonyms for one another. If this mapping commonly appears from one map to another, it is a good practice to save these synonyms in a dictionary file. After being saved, they can be reapplied to another map using automatic mapping.

A dictionary can be created from any existing XSL map and contains all mappings that are not automatically generated by the mapper for the existing map.

To generate and use dictionaries:

1. Create an XSL map that contains specific mappings to reuse in other maps.
2. Go to **Tools > Preferences > XSL Maps > Auto Map** and note the current automatic mapping settings.

Note: Because dictionary entries are dependent upon the current automatic mapping settings, you must make a note of those settings for future use. To later reapply a dictionary mapping, it is best to set the automatic mapping preferences to those that were in effect at the time the dictionary was created. Therefore, it is important to note the automatic mapping settings at the time the dictionary is created.

3. In the XSLT Mapper, right-click in the center panel of the XSLT Mapper and select **Generate Dictionary**.

This prompts you for the dictionary name and the directory in which to place the dictionary.

4. Check the **Open Dictionary** checkbox to view the dictionary after it is created. If the dictionary file is empty, this indicates that no fields were mapped that would not have been mapped with the current automatic mapping settings.
5. To use the dictionary in another map, first load the dictionary by selecting **Tools > Preferences > XSL Maps > Auto Map**.
6. Click **Add** below the **Dictionaries** list.
7. Browse for and select the dictionary XML file that was previously created from a similar map.
8. Click **OK**.
9. Before leaving the automatic mapping preferences, modify the mapping settings to match those used when creating the dictionary.
10. Click **OK**.
11. Perform an automatic mapping of whichever portion of the new map corresponds to the saved dictionary mappings.

For more information about automatic mapping, see [Section 45.3.7, "How to Automatically Map Nodes."](#)

45.3.11 How to Create Map Parameters and Variables

You can create map parameters and variables. You create map parameters in the source tree and map variables in the target tree.

Note the following issues:

- Parameters are created in the source tree, are global, and can be used anywhere in the mappings.
- Variables are created in the target tree, and are either global or local. The location in which they are defined in the target tree determines if they are global or local.
 - Global variables are defined immediately beneath the **<target>** node and immediately above the actual target schema (for example, **POAcknowledge**). Right-click the **<target>** node to create a global variable.
 - Local variables are defined on a specific node beneath the actual target schema (for example, subnode **name** on schema **POAcknowledge**). Local variables can have the same name provided they are in different scopes. Local variables can only be used in their scopes, while global variables can be used anywhere in the mappings.

45.3.11.1 Creating a Map Parameter

To create a map parameter:

1. In the source tree root, right-click and select **Add Parameter**.
The Add Parameter dialog shown in [Figure 45–37](#) appears.
2. Specify details for the parameter. For this example, a parameter named **discount** with a numeric default value of `0.0` is added.

Figure 45–37 Add Parameter Dialog

3. Click **OK**.

45.3.11.2 Creating a Map Variable

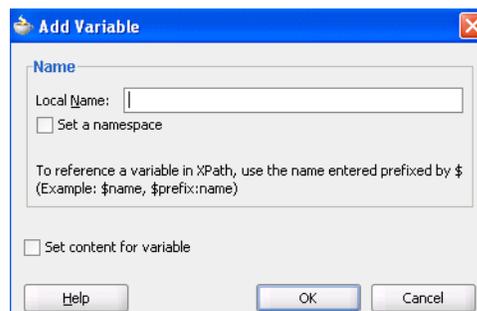
To create a map variable:

1. In the target tree, right-click the target tree root or any node and select **Add Variable**.

The Add Variable dialog shown in [Figure 45–38](#) appears.

2. Specify details.

Since variables appear in the target tree, their XPath expression can be set in the same manner as other XSLT constructs in the target tree after inserting the variable. Therefore, the only required information in this dialog is a name for the variable. If you want to set content for the variable, you must do it through this dialog. Content is handled differently from the XSLT select attribute on the variable.

Figure 45–38 Add Variable Dialog

3. Click **OK**.

The variable is added to the target tree at the position selected.

The variable initially has a warning icon beside it. This indicates that its select XPath statement is undefined. Define the XPath through linking a source node, creating a function, or defining an explicit XPath expression as done for other target elements and XSLT constructs.

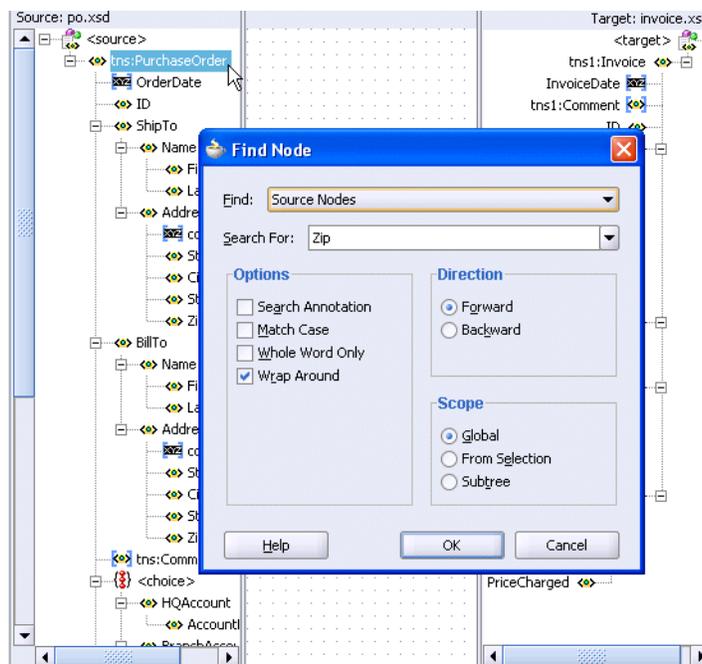
45.3.12 How to Search Source and Target Nodes

You can search source and target nodes. For example, you can search in a source node named **invoice** for all occurrences of the subnode named **price**.

To search source and target nodes:

1. Right-click a source or target node and select **Find** from the context menu.
The Find Node dialog shown in [Figure 45–39](#) is displayed.
2. Enter a keyword for which to search.
3. Specify additional details, as necessary. For example:
 - Select **Search Annotations** if you want annotations text to also be searched.
 - Specify the scope of the search. You can search the entire source or target tree, search starting from a selected position, or search within a selected subtree.

Figure 45–39 Find Node Dialog



The first match found is highlighted, and the Find dialog closes. If no matches are found, a message displays on-screen.

4. Select the **F3** key to find the next match in the direction specified. To search in the opposite direction, select the **Shift** and **F3** keys.

Note: You cannot search on functions or text values set with the **Set Text** option.

45.3.13 How to Control the Generation of Unmapped Target Elements

There are five options for controlling the generation of empty elements in the target XSL:

- Do not generate unmapped nodes (default option).
- Generate empty nodes for *all* unmapped target nodes.
- Generate empty nodes for *all* required, unmapped target nodes.
- Generate empty nodes for *all* nillable, unmapped target nodes.
- Generate empty nodes for *all* required or nillable, unmapped target nodes.

Set these options as follows:

- At the global level:
Select **Tools > Preferences > XSL Maps**. The global setting applies only when a map is created.
- At the map level:
Select **XSL Generation Options** from the map context menu. Each map can then be set independently by setting the options at the map level.

Empty elements are then generated for the selected unmapped nodes. If the unmapped node is nillable, it is generated with `xsi:nil="true"`.

45.3.14 How to Ignore Elements in the XSLT Document

When the XSLT Mapper encounters any elements in the XSLT document that cannot be found in the source or target schema, it cannot process them and displays an `Invalid Source Node Path` error. XSL map generation fails. You can create and import a file that directs the XSLT Mapper to ignore and preserve these specific elements during XSLT parsing by selecting **Preferences > XSL Maps** in the **Tools** main menu of Oracle JDeveloper.

For example, preprocessing may create elements named `myElement` and `myOtherElementWithNS` that you want the XSLT Mapper to ignore when it creates the graphical representation of the XSLT document. You create and import a file with these elements to ignore that includes the following syntax:

```
<elements-to-ignore>
  <element name="myElement"/>
  <element name="myOtherElementWithNS" namespace="NS"/>
</elements-to-ignore>
```

You must restart Oracle JDeveloper after importing the file.

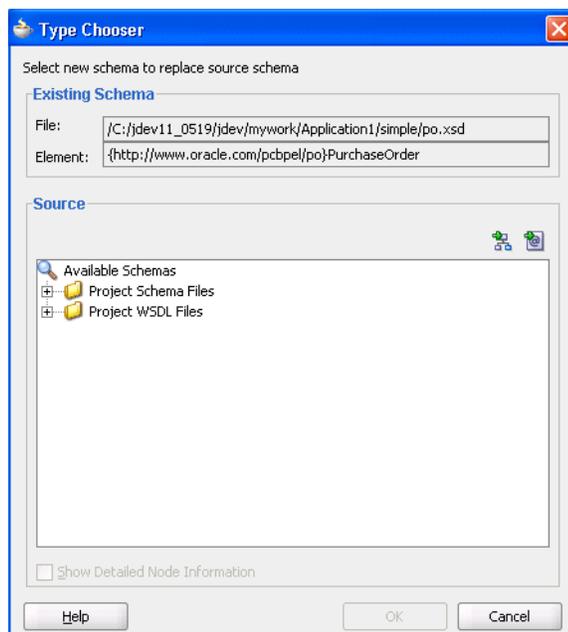
45.3.15 How to Replace a Schema in the XSLT Mapper

You can replace the map source or target schema that currently displays in the XSLT Mapper.

To replace a schema in the XSLT Mapper:

1. In either the source or target panel, right-click and select **Replace Schema**.

This opens the Type Chooser dialog shown in [Figure 45–40](#), which enables you to select the new source or target schema to use.

Figure 45–40 Replacing a Schema

2. Select the replacement schema and click **OK**.
You are then prompted to select if you want to clear expressions in the current map.
3. Select **Yes** or **No**. If expressions are not cleared, you may need to correct the map in source view before reentering design view.

45.3.16 How to Substitute Elements and Types in the Source and Target Trees

You can substitute elements and types in the source and target trees.

Use element substitution when:

- An element is defined as the head of a substitution group in the underlying schema. The element may or may not be abstract. Any element from the substitution group can be substituted for the original element.
- An element is defined as an `any` element. Any global element defined in the schema can be substituted.

Use type substitution when:

- A global type is available in the underlying schema that is derived from the type of an element in the source or target tree. The global type can then be substituted for the original type of the element. Any type derived from an abstract type can be substituted for that abstract type.
- An element in the source or target tree is defined to be of the type `anyType`. Any global type defined in the schema can then be substituted.

Type substitution is supported by use of the `xsi:type` attribute in XML.

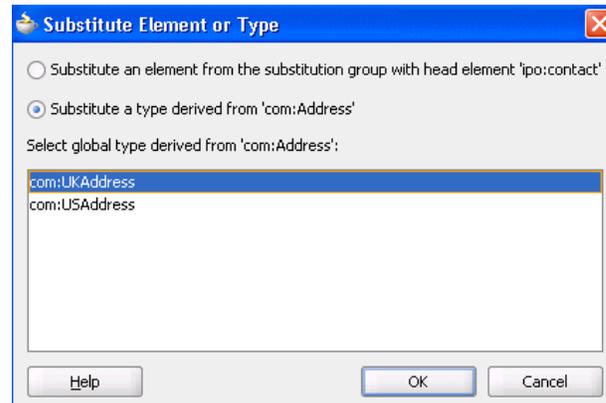
To substitute an element or type in the source and target trees:

1. In the source or target tree, right-click the element for which substitution applies.

- From the context menu, select **Substitute Element or Type**. If this option is disabled, no possible substitutions exist for the element or its type in the underlying schema.

The Substitute Element or Type dialog shown in [Figure 45-41](#) appears.

Figure 45-41 *Substitute Element or Type Dialog*



- Select either **Substitute an element** or **Substitute a type** (only one may be available depending upon the underlying schema).

A list of global types or elements that can be substituted displays in the dialog.

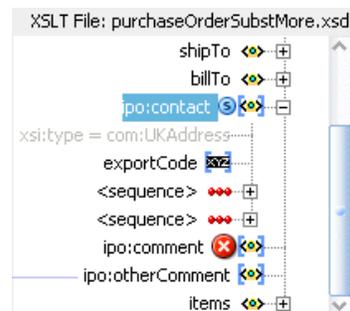
- Select the type or element to substitute.
- Click **OK**.

The element or type is substituted for the originally selected element. This selection displays differently depending upon whether this is a type or element substitution and this is the source or target tree.

- If the element is in the target tree and type substitution is selected:

The **xsi:type** attribute is added beneath the original element, as shown in [Figure 45-42](#). It is disabled in design view and set to the type value that was selected. An **S** icon displays to indicate the node was substituted. You can map to any structural elements in the substituted type.

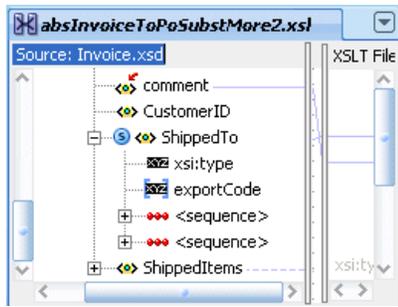
Figure 45-42 *If the Element is in the Target Tree and Type Substitution is Selected*



- If the element is in the source tree and type substitution is selected:

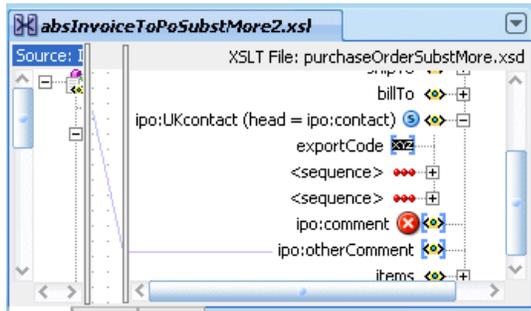
The **xsi:type** attribute is added beneath the original element, as shown in [Figure 45-43](#). An **S** icon is displayed to indicate the node was substituted. You can map from any structural elements in the substituted type.

Figure 45-43 *If the Element is in the Source Tree and Type Substitution is Selected*



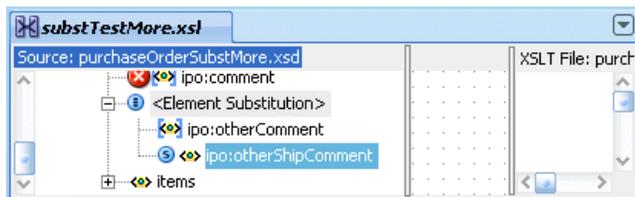
- If the element is in the target tree and element substitution is selected:
The original element is replaced in the tree with the substituted element, as shown in Figure 45-44. A comment indicates the original element name was added and an S icon displays to indicate the node was substituted. You may map to any structural elements in the substituted element.

Figure 45-44 *If the Element is in the Target Tree and Element Substitution is Selected*



- If the element is in the source tree and element substitution is selected:
The original element and its possible replacement both display in the source tree under a new node named **<Element Substitution>**, as shown in Figure 45-45. An S icon displays to indicate the node was added. This feature enables you to build a map where the source object can contain either the original node or a substituted node. You can map to any structural elements in the substituted element.

Figure 45-45 *If the Element is in the Source Tree and Element Substitution is Selected*

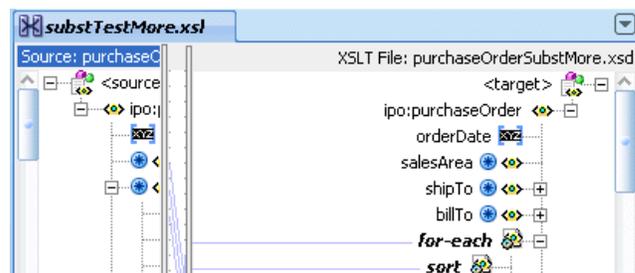


Note: Unlike element substitution, only one type substitution at a time can display in the source tree. This does not prevent you from writing a map that allows the source to sometimes have the original type or the substituted type; you can switch to another type at any time. XPath expressions that map to nodes that may not be visible in the source tree at any given time are still retained.

6. If you want to remove a substituted node, right-click any node with an **S** icon and select **Remove Substitution** from the context menu.
7. If you want to see all possible nodes where substitution is allowed, right-click the source or target tree and select **Show Substitution Node Icons**.

All nodes where substitution is possible are marked with an * icon, as shown in [Figure 45-46](#).

Figure 45-46 All Possible Substitutions

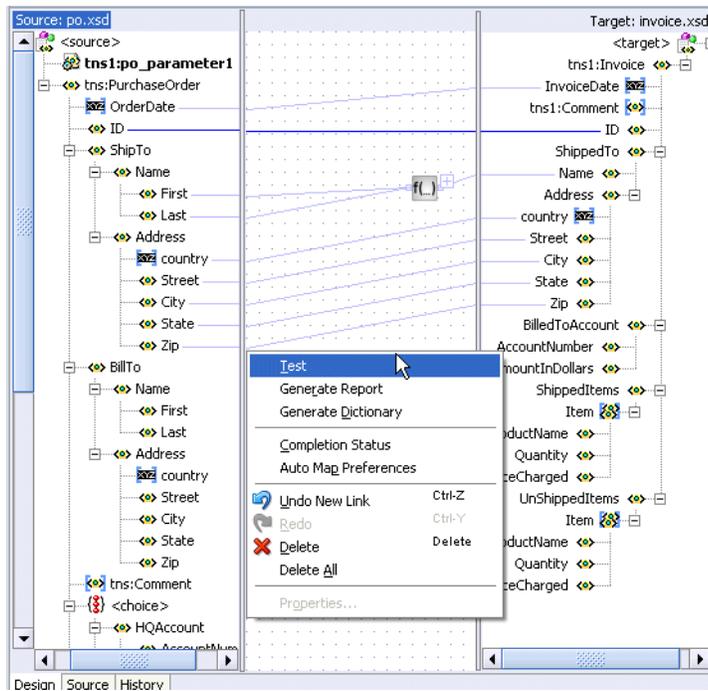


8. To hide the icons, right-click and select **Hide Substitution Node Icons**.

45.4 Testing the Map

The XSLT Mapper provides a test tool to test the style sheet or map. The test tool can be invoked by selecting the **Test** menu item, as shown in [Figure 45-47](#).

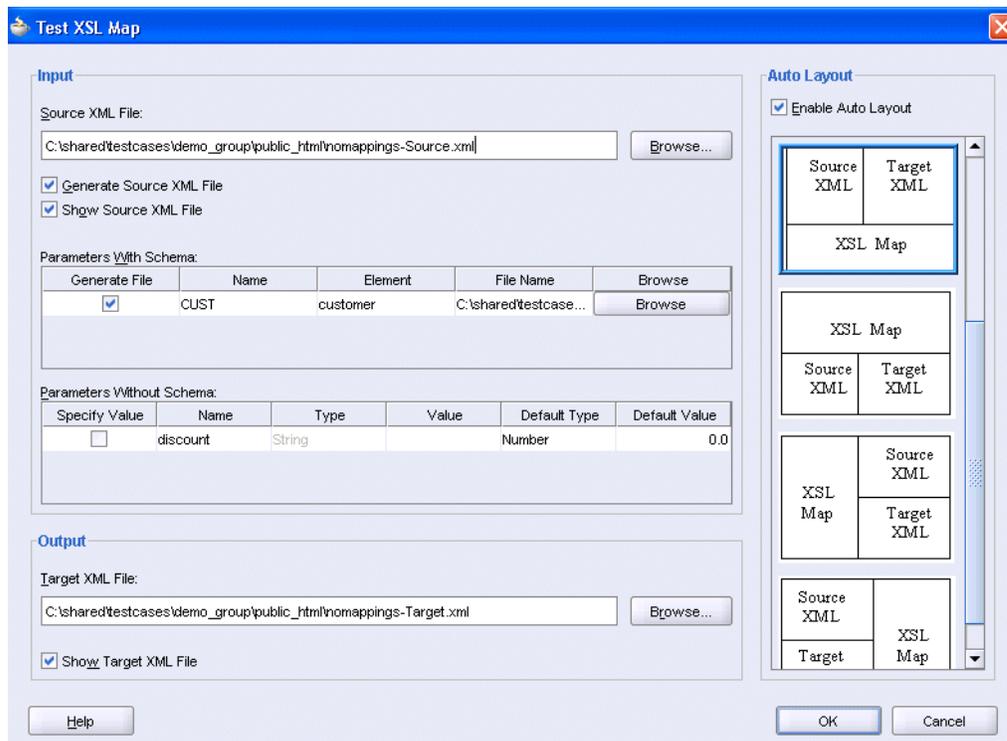
Figure 45-47 Invoking the Test Dialog



45.4.1 How to Test the Transformation Mapping Logic

The Test XSL Map dialog shown in [Figure 45-48](#) enables you to test the transformation mapping logic you designed with the XSLT Mapper. The test settings you specify are stored and do not need to be entered again the next time you test. Test settings must be entered again if you close and reopen Oracle JDeveloper.

Figure 45–48 Test XSL Map Dialog



To test the transformation mapping logic:

1. In the **Source XML File** field, choose to allow a sample source XML file to be generated for testing or click **Browse** to specify a different source XML file.
When you click **OK**, the source XML file is validated. If validation passes, transformation occurs, and the target XML file is created.
If validation fails, no transformation occurs and a message displays on-screen.
2. Select the **Generate Source XML File** checkbox to create a sample XML file based on the map source XSD schema.
3. Select the **Show Source XML File** checkbox to display the source XML files for the test. The source XML files display in an Oracle JDeveloper XML editor.

If the map has defined parameters, the **Parameters With Schema** or **Parameters Without Schema** tables can appear.

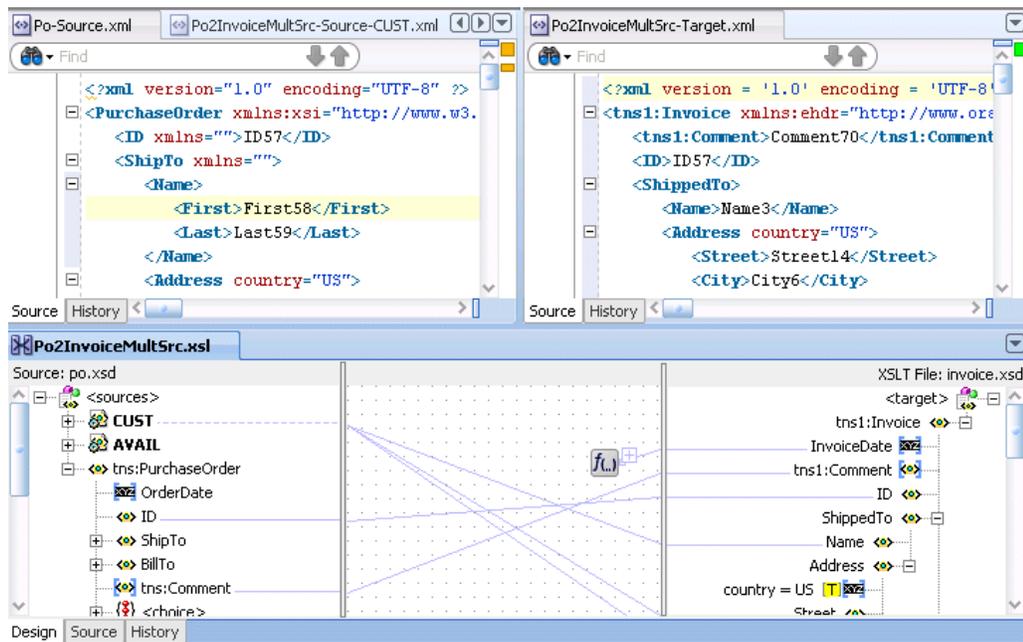
- a. If the **Parameters With Schema** table appears, you can specify an input XML file for the parameter using the **Browse** button. Select the **Generate File** checkbox if you want to generate a file.
- b. If the **Parameters Without Schema** table appears, you can specify a value by selecting the **Specify Value** checkbox and making appropriate edits to the **Type** and **Value** columns.
4. In the **Target XML File** field, enter a file name or browse for a file name in which to store the resulting XML document from the transformation.
5. Select the **Show Target XML File** checkbox to display the target XML file for the test. The target XML file displays in an Oracle JDeveloper XML editor.

6. If you select to show both the source and target XML, you can customize the layout of your XML editors. Select **Enable Auto Layout** in the upper right corner and click one of the patterns.
7. Click **OK**.

The test results shown in [Figure 45–49](#) appear.

For this example, the source XML and target XML display side-by-side with the XSL map underneath (the default setting). Additional source XML files corresponding to the **Parameters With Schema** table are displayed as tabs in the same area as the main source file. You can right-click an editor and select **Validate XML** to validate the source or target XML against the map source or target XSD schema.

Figure 45–49 Test Results



Note: If the XSL map file contains domain value map (DVM) and XRef XPath functions, it cannot be tested. These functions cannot be executed during design time; they can only be executed during runtime.

45.4.2 How to Generate Reports

You can generate an HTML report with the following information:

- XSL map file name, source and target schema file names, their root element names, and their root element namespaces
- Target document mappings
- Target fields not mapped (including mandatory fields)
- Sample transformation map execution

Follow these instructions to generate a report.

1. In the center panel, right-click and select **Generate Report**.

The Generate Report dialog appears, as shown in [Figure 45–50](#). Note that if the map has defined parameters, the appropriate parameter tables appear.

Figure 45–50 The Generate Report Dialog

Generate Report

File Name:
nomappings-Report.html

Directory Name:
C:\shared\tstcases\demo_group\public_html

Input

Source XML File:
C:\shared\tstcases\demo_group\public_html\nomappings-Source.xml

Generate Source XML File

Parameters With Schema:

Generate File	Name	Element	File Name	Browse
<input checked="" type="checkbox"/>	CUST	customer	C:\shared\tstcases...	Browse

Parameters Without Schema:

Specify Value	Name	Type	Value	Default Type	Default Value
<input type="checkbox"/>	discount	String		Number	0.0

Open Report
 Add To Project

Help OK Cancel

For more information about the fields, see the online Help for the Generate Report dialog.

45.4.2.1 Correcting Memory Errors When Generating Reports

If you attempt to generate a report and receive an out-of-memory error, increase the heap size of the JVM as follows.

To increase the JVM heap size:

1. Open the `JDev_Oracle_Home\jdev\bin\jdev.conf` file.
2. Go to the following section:

```
# Set the maximum heap to 512M
#
AddVMOption    -Xmx512M
```

3. Increase the size of the heap as follows (for example, to 1024):

```
AddVMOption    -Xmx1024M
```

In addition, you can also unselect the **Open Report** option on the Generate Report dialog before generating the report.

45.4.3 How to Customize Sample XML Generation

You can customize sample XML generation by specifying the following parameters. Select **Preferences > XSL Maps** in the **Tools** main menu of Oracle JDeveloper to display the Preferences dialog.

- Number of repeating elements
Specifies how many occurrences of an element are created if the element has the attribute `maxOccurs` set to a value greater than 1. If the specified value is greater than the value of the `maxOccurs` attribute for a particular element, the number of occurrences created for that particular element is the `maxOccurs` value, not the specified number.
- Generate optional elements
If selected, any optional element (its attribute `minOccurs` set to a value of 0) is generated the same way as any required element (its attribute `minOccurs` set to a value greater than 0).
- Maximum depth
To avoid the occurrence of recursion in sample XML generation caused by optional elements, specify a maximum depth in the XML document hierarchy tree beyond which no optional elements are generated.

45.5 Demonstrating the New Features of the XSLT Mapper

This sample demonstrates the following new features of the XSLT mapper:

- Element and type substitution
- Multiple sources use
- New XSL constructs `xsl:sort` and `xsl:copy-of`
- New variable use

In addition to this sample, Oracle provides other transformation samples that are available for download from the Oracle Technology Network (OTN). These samples are described in [Table 45-2](#). To access these samples, visit the following URL:

http://www.oracle.com/technology/sample_code/products/soa

Table 45-2 Transformation Samples

Sample	Description
mapper-101-basic-mapping	Demonstrates creation and basic editing of an XSLT map.
mapper-102-import-and-test	Demonstrates the following XSL mapper features: <ul style="list-style-type: none"> ■ Import of external XSL ■ Test execution with an overview of XML editor validation ■ Report execution
mapper-104-auto-mapping	Demonstrates the automatic mapping feature of the XSLT Editor.
mapper-105-multiple-sources	Demonstrates the use of multiple sources. The following topics are also covered in the process of creating the map sample. <ul style="list-style-type: none"> ■ Inserting a cloned <code>for-each</code> ■ Adding predicates to XPath expressions ■ Using variables

Table 45–2 (Cont.) Transformation Samples

Sample	Description
mapper-107-extension-functions	Demonstrates the use of user-defined extension functions.
mapper-108-substitution-mapping	Demonstrates the use of element substitution when: <ul style="list-style-type: none"> ■ An element is defined as the head of a substitution group in the underlying schema. The element may or may not be abstract. Any element from the substitution group can be substituted for the original element. ■ An element is defined as an any element. Any global element defined in the schema can be substituted for the any element. This is subject to any namespace constraints placed on the definition of the any element.
mapper-109-whats-new	Demonstrates the new features in the XSLT Mapper. These features are described in Section 45.5.1, "Opening the Application" through Section 45.5.7, "Testing the Map."

45.5.1 Opening the Application

You first create the sample application. When complete, the application matches the one provided in the `WhatsNewApplication` directory described in Step 1.

1. Download sample `mapper-109-whats-new` from OTN.

The sample includes the following files and directories:

- `artifacts/schemas/po.xsd` and `Attachment.xsd`: source schema
 - `artifacts/schemas/invoice.xsd` and `ReasonCodes.xsd`: target schema
 - `artifacts/application`: starting application for this sample
 - `WhatsNewApplication` directory: completed sample map
2. Copy the `artifacts/application` folder to a separate work area.
 3. Start Oracle JDeveloper.
 4. Click `WhatsNewApplication.jws` in the `artifacts/application` folder you copied to a separate area.
 5. If prompted to migrate files, click **Yes**.

The **WhatsNewApplication** displays in the Application Navigator.

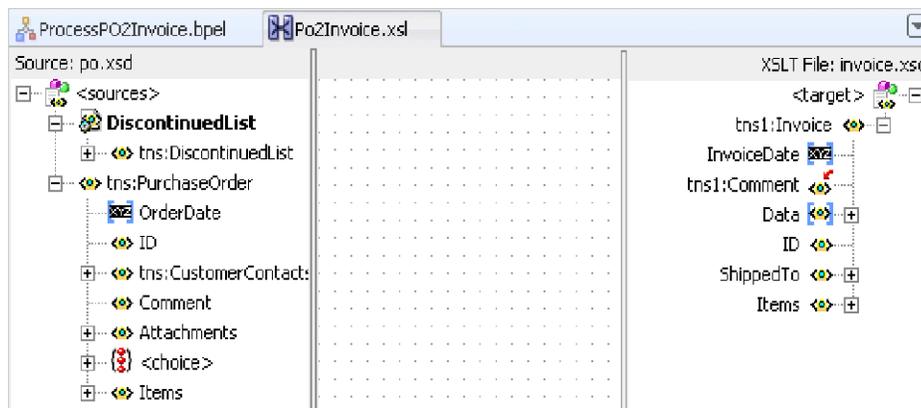
45.5.2 Creating a New XSLT Map in the BPEL Process

You now create a new XSLT map with two sources that is invoked from the BPEL process included in the **WhatsNewApplication** application.

1. In the Application Navigator, double-click the **ProcessPO2Invoice.bpel** BPEL process.
2. From the **BPEL Activities and Components** section of the Component Palette, drag a **Transform** activity below the **SetDiscontinuedProducts** assign activity.
3. Double-click the **Transform** activity.
4. In the **Name** field of the **General** tab, enter `Po2Invoice`.
5. In the **Transformation** tab, perform the following steps:

- a. Click the **Add** icon.
- b. From the **Source Variable** list, select **inputVariable**.
- c. From the **Source Part** list, select **payload**.
This variable contains the purchase order that is input to the BPEL process.
- d. Click **OK**.
- e. Click the **Add** icon a second time and select **DiscontinuedList** from the **Source Variable** list. The variable is created in the **SetDiscontinuedProducts** assign activity before the transformation activity.
- f. Click **OK**.
- g. From the **Target Variable** list, select **outputVariable**. This is the invoice that is returned from the BPEL process.
- h. In the **Mapper File** field, change the name to `xsl/Po2Invoice`.
- i. Click the **Create Mapping** icon to the right of the **Mapper Name** field to create and open the mapper file.
The XSLT Mapper opens.
- j. From the **File** menu, select **Save All**. Your map looks as shown in [Figure 45–51](#). Note that the second source is loaded as a parameter with the name **DiscontinuedList**:

Figure 45–51 XSLT Mapper File



45.5.3 Using Type Substitution to Map the Purchase Order Items

You now use type and element substitutions to map the purchase order (PO) items to the invoice items.

1. In the target tree, expand the tree so that **Invoice/Items/Item** is visible. Note that the **Item** element has an error icon next to it.
2. Move the mouse over the element to display a tool tip indicating that this element is defined as an abstract type.

To map to the **Item** element, you must first indicate which type the element takes in the final XML output.

3. Perform the following steps to indicate what type the element takes:
 - a. Right-click the **Item** element and select **Substitute Element or Type**.

The Substitute Element or Type dialog appears.

- b. Select **ShippedItemType** from the list and click **OK**.

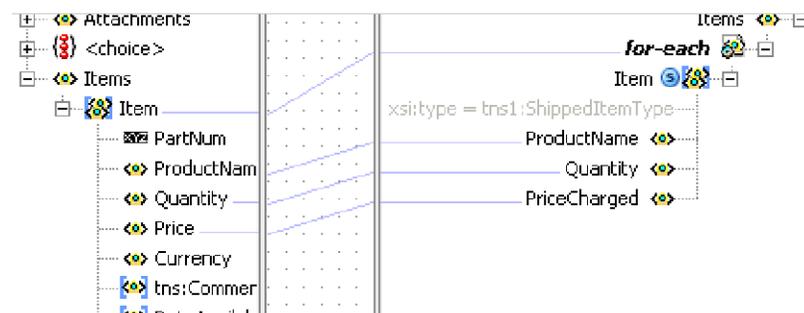
The **Item** element structure is filled out. The `xsi:type` attribute sets the type of the **Item** element in the target XML.

Note: If you view `invoice.xsd`, note that **ShippedItemType** is derived from the abstract type **ItemType**, which is the type of the **Item** element.

4. Drag **PurchaseOrder/Items** to **Invoice/Items** to invoke the automatic mapper to map these nodes. To review automatic mapping functionality, see [sample mapper-104-auto-mapping](#).

When complete, the **Item** elements in your map now look as shown in [Figure 45–52](#):

Figure 45–52 Item Elements in XSLT Mapper



5. From the **File** menu, select **Save All** to save the map file.

45.5.4 Referencing Additional Source Elements

You now use the information in the additional source variable, **DiscontinuedList**, to eliminate items that have been discontinued. If the product name for an item is in **DiscontinuedList**, then that item cannot be shipped and is not placed in the final shipped item list.

1. Add an **if** statement above the **Item** node in the target tree by right-clicking the **Item** node and selecting **Add XSL Node > if**.

The **if** statement must test if a discontinued product exists in **DiscontinuedList** with the name of the current item. The item is added only to the shipped items if it is not in **DiscontinuedList**. There are many ways to define the test expression for the **if** statement. One way is described in the following steps.

2. Define the test expression for the **if** statement by selecting the following (note that the method for how variables are set has changed from the previous version of Oracle JDeveloper):

- a. Add a global variable to the target tree by right-clicking the **Invoice** node and selecting **Add Variable**.

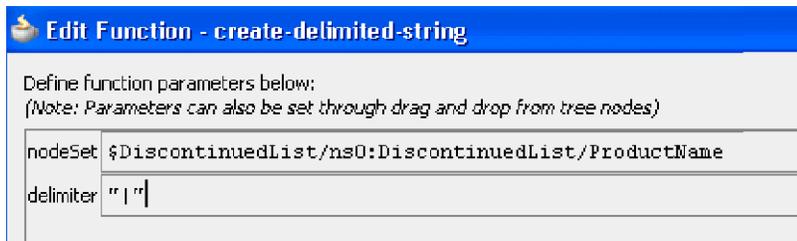
The Add Variable dialog appears.

- b. In the **Local Name** field, enter `DelimitedList`. In the following steps, this variable is set to a string with a delimited list of discontinued product names.

- c. Click **OK**.
The variable is added with a warning icon next to it.
- d. To set the value of the variable, drag the **create-delimited-string** function from the **String** section of the Component Palette to the center panel.
- e. Drag **DiscontinuedList/ProductName** to the input side of the **create-delimited-string** function.
- f. Drag the output side of the **create-delimited-string** function to the new variable named **DelimitedList**.
- g. Double-click the **create-delimited-string** function to open the Edit Function dialog.
- h. In the **delimiter** field, add the pipe character.
- i. Click **OK**.

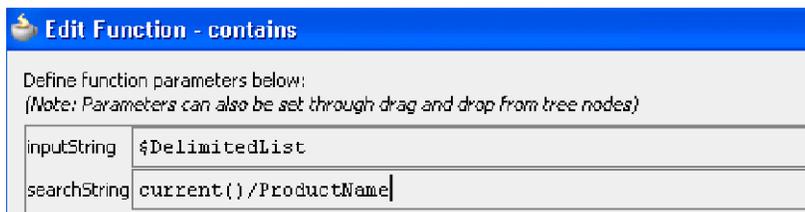
Note that the input source is referenced in XPath expressions with **\$DiscontinuedList**, as shown in [Figure 45–53](#). This source is referenced as an input parameter in XPath expressions.

Figure 45–53 *\$DiscontinuedList*



3. To set the XPath expression for the **if** statement, drag the **contains** function from the **String** section of the Component Palette to the center panel.
4. Drag the **not** function from the **Logical Functions** section of the Component Palette to the shaded area surrounding the **contains** function you added in Step 3.
5. Drag a line from the output side of the **contains** function to the input side of the **not** function.
6. Drag a line from the output side of the **not** function to the **if** statement.
7. Double-click the **contains** function to open the Edit Function dialog.
8. Enter values for the **inputString** and **searchString**, as shown in [Figure 45–54](#), and click **OK**.

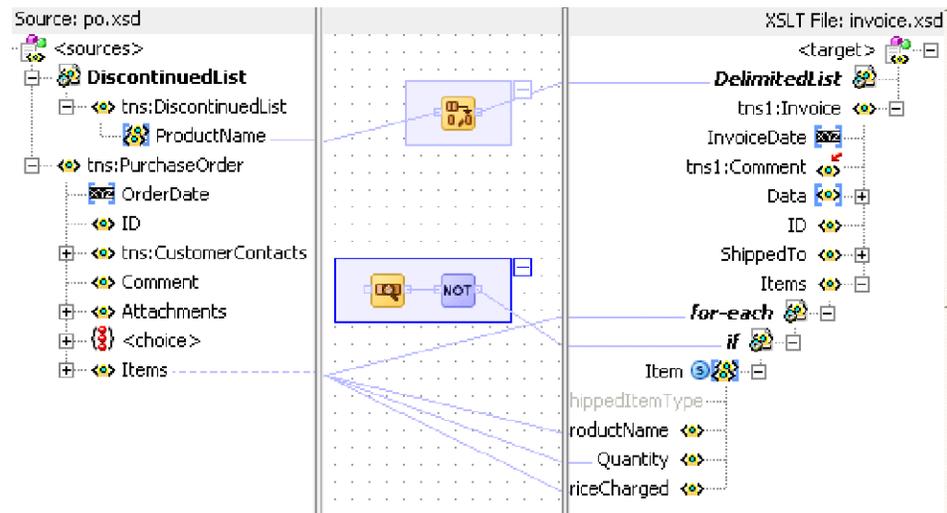
Figure 45–54 *Edit Function Dialog*



9. From the **File** menu, select **Save All** to save the map file.

The map file now looks as shown in [Figure 45–55](#).

Figure 45–55 Mapper File



45.5.5 Using Element Substitution to Map the Shipping Address

You now map a substituted shipping contact element in the source to the **ShippedTo** element in the target.

1. Expand the **PurchaseOrder/CustomerContacts** element in the source to see the **Contact** element.

Note that this element has an error icon next to it.

2. Place the mouse over this element to display a tool tip indicating that this element is abstract.

In this situation, you must perform an element substitution to map the element.

3. Right-click the **Contact** element in the source tree and select **Substitute Element or Type**.

The Substitute Element or Type dialog is displayed with a list of elements in the substitution group of the abstract element **Contact**.

4. Select **ShipToContact** and click **OK**.

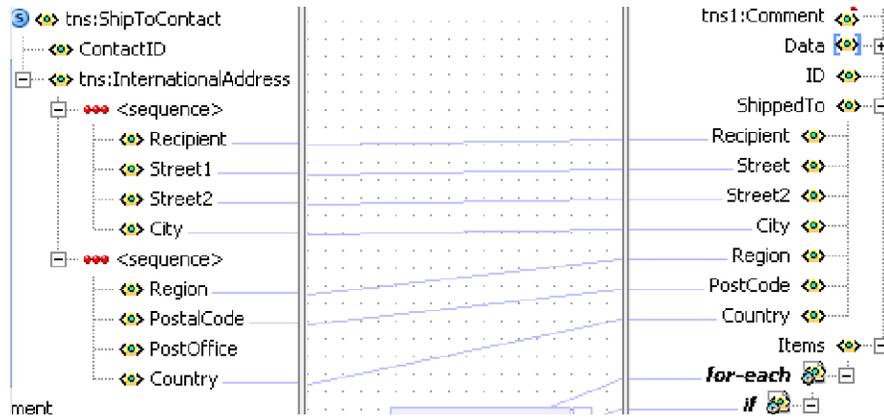
This is the element that you expect in the input XML. The structure of the **ShipToContact** element is now displayed in the source tree.

5. Expand the **ShipToContact/InternationalAddress** element in the source tree to show the address fields.
6. Expand the **ShippedTo** element in the target tree to show the target address fields.

Note the similarity in field names here, indicating that the automatic mapper can be used.

7. Drag the **InternationalAddress** element in the source tree to the **ShippedTo** element in the target tree and use the automatic mapper to help map the address fields below these elements.
8. Map any remaining elements not matched by the automatic mapper so that this section of the map is as shown in [Figure 45–56](#):

Figure 45–56 XSLT Mapper



9. From the File menu, select **Save All** to save the map file.

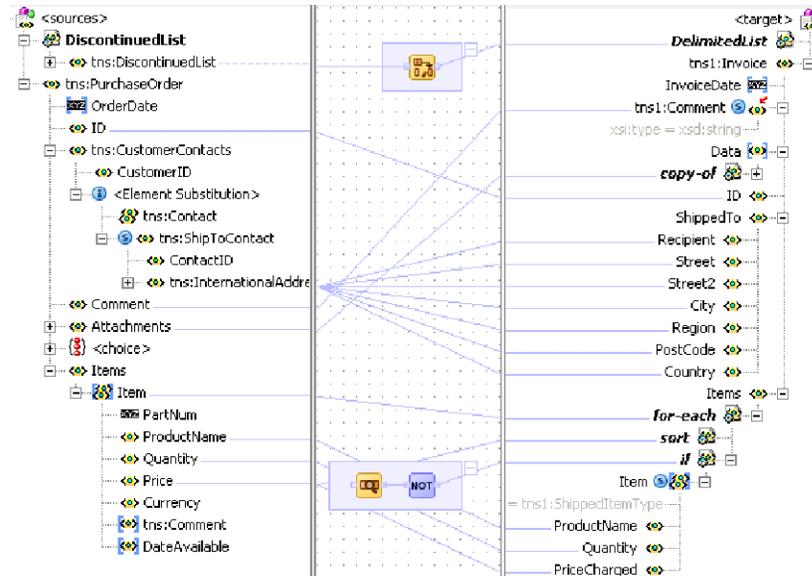
45.5.6 Mapping the Remaining Fields

1. Map **PurchaseOrder/ID** to **Invoice/ID**.
2. Expand **Invoice/Data** to show an **any** element.
3. Use the **copy-of xsl** statement to copy the attachment data from the source to the target **any** element:
 - a. Right-click the **Invoice/Data/any** element and select **Add XSL Node > copy-of**.
 The **copy-of** statement is added and the original **any** element is grayed out. This indicates that it is to be replaced by the nodes selected by the **copy-of** statement.
 - b. To set the **copy-of** selection, drag the **PurchaseOrder/Attachments** element in the source tree to the **copy-of** statement.
4. Perform the following steps to map the **PurchaseOrder/Comment** field to the **Invoice/Comment** field. Note that the **Invoice/Comment** field is an **anyType** element.
 - a. Right-click the **Invoice/Comment** field and select **Substitute Element or Type**.
 - b. Select **xsd:string** from the list of types provided.
 - c. Drag the **PurchaseOrder/Comment** field to the **Invoice/Comment** field to map the fields.
5. Add an XSL **sort** statement to the **for-each** statement:
 - a. Right-click the **for-each** statement in the target tree and select **Add XSL Node > sort**.
 The Sort Edit dialog appears.
 - b. Select **sort according to data-type Number**.
 - c. Select **sort order Descending**.
 - d. Click **OK**. The sort node is added to the target tree.
 - e. Drag **PurchaseOrder/Items/Item/Price** from the source tree to the sort node in the target tree.

This sets the field on which to sort.

- From the **File** menu, select **Save All** to save the map file. The map now looks as shown in [Figure 45–57](#):

Figure 45–57 XSLT Mapper



45.5.7 Testing the Map

An XSL map can be tested independently from the BPEL process in Oracle JDeveloper using the XSLT Mapper test tool. XML files can be input for each source input to the map.

- Right-click the center panel and select **Test**.

The Test XSL Map dialog appears after a warning dialog. The warning indicates that you can test the map by creating your own sample input XML. The sample XML generator cannot generate sample data for the source tree substitutions.

A sample input XML file is provided: **artifacts/xml/POInput.xml**.

- Follow these steps to select the sample input file for testing:
 - Uncheck the **Generate Source XML File** checkbox.
 - Click the **Browse** button for the **Source XML File** field.
 - Navigate to select the **artifacts/xml/POInput.xml** file.

A second sample file has been created with discontinued item data. This file is **artifacts/xml/DiscontinuedItems.xml**.

- Follow these steps to use this file as the second source input.
 - Uncheck the **Generate File** checkbox to the left of the **DiscontinuedList** parameter name in the **Parameters With Schema** section of the dialog.
 - Click **Browse** for the **DiscontinuedList** parameter and select the **artifacts/xml/DiscontinuedItems.xml** file.
- Click **OK** on the Test XSL Mapper dialog to run the test.

A **PO2Invoice-Target.xml** file is generated by the execution of the map. Note the use of **xsi:type** attributes, the **Attachments** node created by the **copy-of** statement, and the ordering of items caused by the **sort** statement in the **PO2Invoice-Target.xml** file.

Working with Domain Value Maps

This chapter describes how to use domain value maps to map the vocabulary used by different domains.

This chapter includes the following sections:

- [Section 46.1, "Introduction to Domain Value Maps"](#)
- [Section 46.2, "Creating Domain Value Maps"](#)
- [Section 46.3, "Editing a Domain Value Map"](#)
- [Section 46.4, "Using Domain Value Map Functions"](#)
- [Section 46.5, "Creating a Domain Value Map Use Case for Hierarchical Lookup"](#)
- [Section 46.6, "Creating a Domain Value Map Use Case For Multiple Values"](#)

46.1 Introduction to Domain Value Maps

Domain value maps operate on actual data values that transit through the infrastructure at runtime. They enable you to map from one vocabulary, used in a given domain, to another vocabulary used in a different domain. For example, one domain might represent a city with a long name (`Boston`) while another domain may represent a city with a short name (`BO`). In such cases, you can directly map the values by using domain value maps. A direct mapping of values between two or more domains is known as point-to-point mapping. [Table 46–1](#) shows a point-to-point mapping for cities between two domains:

Table 46–1 *Point-to-Point Mapping*

CityCode	CityName
BELG_MN_STLouis	BelgradeStLouis
BELG_NC	BelgradeNorthCarolina
BO	Boston
NP	Northport
KN_USA	KensingtonUSA
KN_CAN	KensingtonCanada

Each domain value map typically holds a specific category of mappings among multiple applications. For example, one domain value map might hold mappings for city codes and another might hold mappings for state codes.

Domain value map values are static. You specify the domain value map values at design time using Oracle JDeveloper, and then at runtime, the domain value map columns are looked up for values.

Note: To dynamically integrate values between applications, you can use the Cross referencing feature of Oracle SOA Suite. For information about cross references, see [Chapter 47, "Working with Cross References"](#)

46.1.1 Domain Value Map Features

The domain value map functionality consists of the following features:

- [Section 46.1.1.1, "Qualifier Support"](#)
- [Section 46.1.1.2, "Qualifier Order Support"](#)
- [Section 46.1.1.3, "One-to-Many Mapping Support"](#)

46.1.1.1 Qualifier Support

Qualifiers qualify mappings. A mapping may not be valid unless qualified with additional information. For example, a domain value map containing city code to city name mapping may have multiple mappings from KN to Kensington because Kensington is a city in Canada as well as in USA. So, this mapping requires a qualifier (USA or Canada) to qualify when the mapping becomes valid, as shown in [Table 46–2](#).

Table 46–2 *Qualifier Support Example*

Country (Qualifier)	CityCode	CityName
USA	BO	Boston
USA	BELG_NC	Belgrade
USA	BELG_MN_Streams	Belgrade
USA	NP	Northport
USA	KN	Kensington
Canada	KN	Kensington

You can also specify multiple qualifiers for a domain value map. For example, as shown in [Table 46–3](#), BELG to Belgrade mapping can also be qualified with state name.

Table 46–3 *Multiple Qualifier Support Example*

Country (Qualifier)	State (Qualifier)	CityCode	CityName
USA	Massachusetts	BO	Boston
USA	North Carolina	BELG	Belgrade
USA	Minnesota	BELG	Belgrade
USA	Alabama	NP	Northport
USA	Kansas	KN	Kensington
Canada	Prince Edward Island	KN	Kensington

Qualifiers are used only to qualify the mappings. So, the qualifier values cannot be looked up.

46.1.1.2 Qualifier Order Support

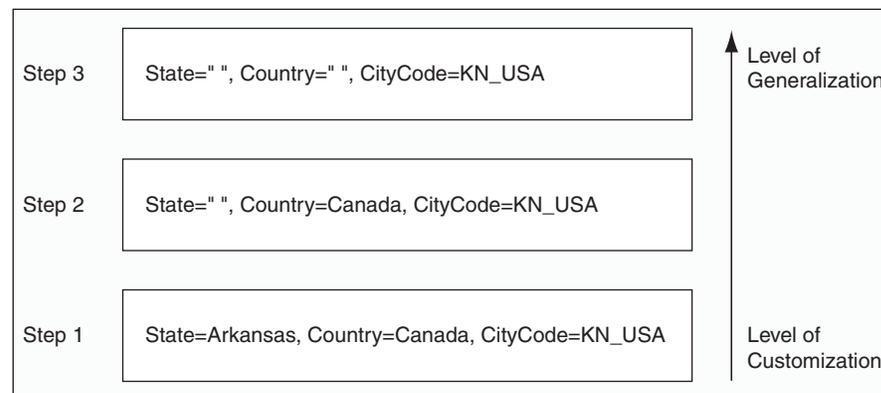
A qualifier order is used to find the best match during lookup at runtime. The order of a qualifier varies from highest to lowest depending on the role of the qualifier in defining a more exact match. In [Table 46-3](#), the state qualifier can have a higher order than the country qualifier, as a matching state indicates a more exact match.

Domain value maps support hierarchical lookup. If you specify a qualifier value during a lookup and no exact match is found, then the lookup mechanism tries to find a more generalized match by setting the higher order qualifiers to a " ". It proceeds until a match is found, or until a match is not found with all qualifiers set to a " ". [Figure 46-1](#) describes hierarchical lookup performed for the following lookup on [Table 46-3](#):

State=Arkansas, Country=Canada, CityCode=KN_USA

In this example, the State qualifier has a qualifier value of 1 and the Country qualifier has a qualifier value of 2.

Figure 46-1 Hierarchical Lookup Example



As shown in [Figure 46-1](#), the lookup mechanism sets the higher order qualifier STATE to the exact lookup value Arkansas and uses Canada | " " for the lower order qualifier Country.

When no match is found, the lookup mechanism sets the higher order qualifier, STATE to value " " and sets the next higher qualifier Country to an exact value Canada.

When no match is found, the lookup mechanism sets the value of the previous higher order qualifier Country to value " ". One matching row is found where CityCode is KN_USA and Kensington is returned as value.

[Table 46-4](#) provides a summary of these steps.

Table 46-4 Domain Value Map Lookup Result

STATE	COUNTRY	Short Value	Lookup Result
Arkansas	CANADA " "	KN_USA	No Match
" "	CANADA	KN_USA	No Match
" "	" "	KN_USA	Kensington

46.1.1.3 One-to-Many Mapping Support

You can map one value to multiple values in a domain value map. For example, a domain value map for Payment Terms can contain mapping of payment terms to three values, such as discount percentage, discount period, and total payment period, as shown in [Table 46–5](#).

Table 46–5 One-to-Many Mapping Support

Payment Term	Discount Percentage	Discount Period	Net Credit Period
GoldCustomerPaymentTerm	10	20	30
SilverCustomerPaymentTerm	5	20	30
RegularPaymentTerm	2	20	30

46.2 Creating Domain Value Maps

You can create one or more domain value maps in a SOA Composite application of Oracle JDeveloper, and then at runtime, use it to look up for column values.

46.2.1 How to Create Domain Value Maps

You can create a domain value map by using the Create Domain Value Map(DVM) File dialog in Oracle JDeveloper.

To create a domain value map:

1. In the Application Navigator, right-click the project in which you want to create a domain value map and select **New**.

The New Gallery dialog is displayed.

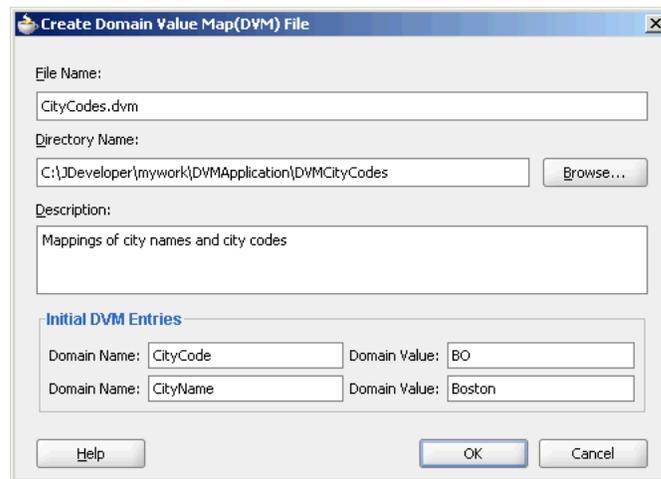
2. Expand the **SOA Tier** node, and then select the **Transformations** category.
3. In the Items list, select **Domain Value Map(DVM)** and click **OK**.

The Create Domain Value Map(DVM) File dialog is displayed.

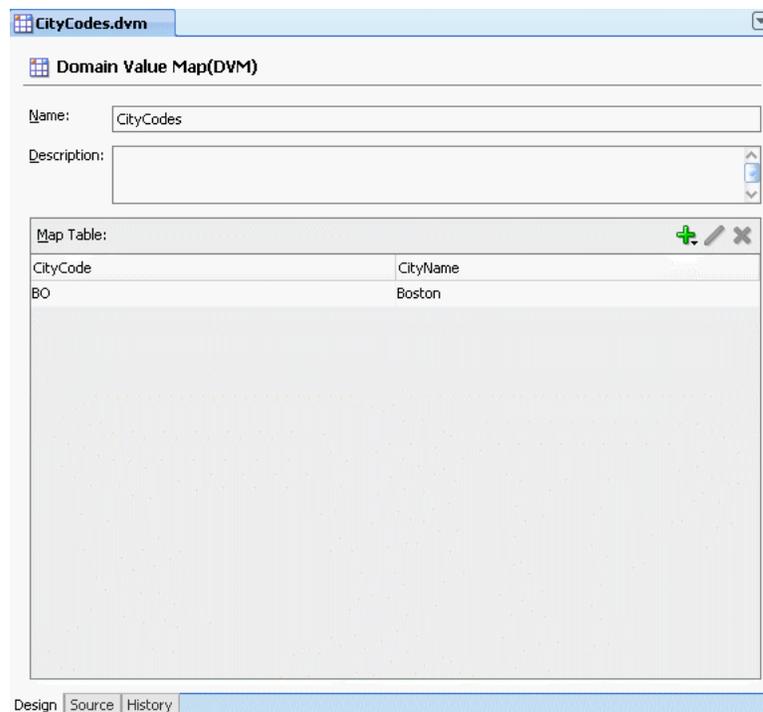
4. In the **File Name** field, enter the name of the domain value map file. For example, specify `CityCodes` to identify a domain value map for city names and city codes.
5. In the **Description** field, enter a description for the domain value map. For example, `Mappings of city names and city codes`. This field is optional.
6. In the **Domain Name** field, enter a name for each domain. For example, you can enter `CityCode` in one **Domain Name** field and `CityName` in another. Each domain name must be unique in a domain value map.

Note: You can later add more domains to a domain value map by using the Domain Value Map Editor.

7. In the **Domain Value** field, enter a value corresponding to each domain. For example, enter `BO` for `CityCode` domain and `Boston` for `CityName` domain as shown in [Figure 46–2](#).

Figure 46–2 Populated Create Domain Value Map File Dialog**8.** Click **OK**.

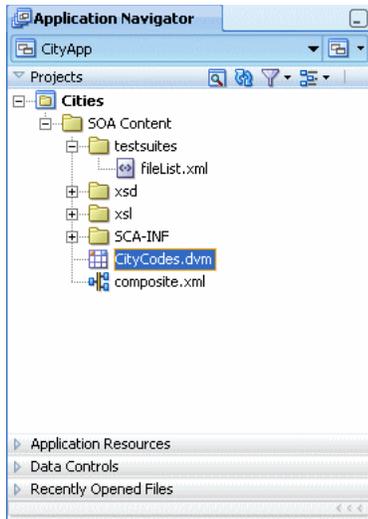
The Domain Value Map Editor is displayed, as shown in [Figure 46–3](#).

Figure 46–3 Domain Value Map Editor

46.2.2 What Happens When You Create a Domain Value Map

A file with extension `.dvm` gets created and appears in the Application Navigator, as shown in [Figure 46–4](#).

Figure 46–4 A Domain Value Map File in Application Navigator



All .dvm files are based on the following schema definition (XSD) file:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- Copyright (c) 2006, Oracle. All rights reserved. -->
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://xmlns.oracle.com/dvm"
            xmlns:tns="http://xmlns.oracle.com/dvm"
            elementFormDefault="qualified"
            attributeFormDefault="unqualified">

  <xsd:element name="dvm">
    <xsd:annotation>
      <xsd:documentation>The Top Level Element
      </xsd:documentation>
    </xsd:annotation>
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="description" minOccurs="0" type="xsd:string">
          <xsd:annotation>
            <xsd:documentation>The DVM Description. This is optional
            </xsd:documentation>
          </xsd:annotation>
        </xsd:element>
        <xsd:element name="columns">
          <xsd:annotation>
            <xsd:documentation>This element holds DVM's column List.
            </xsd:documentation>
          </xsd:annotation>
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="column" minOccurs="2" maxOccurs="unbounded">
                <xsd:annotation>
                  <xsd:documentation>This represents a DVM Column
                  </xsd:documentation>
                </xsd:annotation>
                <xsd:complexType>
                  <xsd:attribute name="name" use="required" type="xsd:string"/>
                  <xsd:attribute name="qualifier" default="false"
                    type="xsd:boolean"
                    use="optional"/>
                </xsd:complexType>
              </xsd:element>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

```

        <xsd:attribute name="order" use="optional"
type="xsd:positiveInteger"/>
    </xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
<xsd:element name="rows" minOccurs="0">
    <xsd:annotation>
        <xsd:documentation>This represents all the DVM Rows.
    </xsd:documentation>
    </xsd:annotation>
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="row" minOccurs="1" maxOccurs="unbounded">
                <xsd:annotation>
                    <xsd:documentation>
                        Each DVM row of values
                    </xsd:documentation>
                </xsd:annotation>
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element name="cell" minOccurs="2" maxOccurs="unbounded"
                            type="xsd:string">
                            <xsd:annotation>
                                <xsd:documentation>This is the value for this row and for
each column in the same order as defined in Columns.
                            </xsd:documentation>
                            </xsd:annotation>
                        </xsd:element>
                    </xsd:sequence>
                </xsd:complexType>
            </xsd:element>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
</xsd:sequence>
    <xsd:attribute name="name" use="required" type="xsd:string"/>
</xsd:complexType>
</xsd:element>
<xsd:annotation>
    <xsd:documentation>This Schema is used to validate the DVM Document got for
creation and
update of a DVM.
    </xsd:documentation>
</xsd:annotation>
</xsd:schema>

```

46.3 Editing a Domain Value Map

After you have created a domain value map, you can edit it and make adjustments to the presentation of data in the Domain Value Map Editor.

46.3.1 Adding Columns to a Domain Value Map

A domain value map column defines the domain whose values you want to map with other domains.

To add a column to a domain value map:

1. Click **Add**.
2. Select **Add Column**.
The Create DVM Column dialog is displayed.
3. In the **Name** field, enter a column name.
4. In the **Qualifier** field, select **True** to set this column as a qualifier, else select **False**.
5. In the **Qualifier Order** field, enter a qualifier number. This field is enabled only if you have selected True in the **Qualifier** field.
6. Click **OK**.

46.3.2 Adding Rows to a Domain Value Map

A domain value map row contains the values of the domains.

To add a row to a domain value map:

1. In the Domain Value Map Editor, click **Add**.
2. Select **Add Row**.

46.4 Using Domain Value Map Functions

After creating a domain value map, you can use the XPath functions of the domain value map to look up for appropriate values and populate the targets for the applications at runtime.

46.4.1 Understanding Domain Value Map Functions

You can use the `dvm:lookupValue` and `dvm:lookupValue1M` XPath functions to look up a domain value map for a single value or multiple values at runtime.

46.4.1.1 `dvm:lookupValue`

The `dvm:lookupValue` function returns a string by looking up the value for the target column in a domain value map, where the source column contains the given source value.

- Usage 1

```
dvm:lookupValue(dvmMetadataURI as string, SourceColumnName as string,  
SourceValue as string, TargetColumnName as string, DefaultValue as string) as  
string
```

Example:

```
dvm:lookupValue('cityMap.dvm', 'CityCodes', 'BO', 'CityNames', 'CouldNotBeFound')
```

- Usage 2

```
dvm:lookupValue(dvmMetadataURI as string, SourceColumnName as string,  
SourceValue as string, TargetColumnName as string, DefaultValue as string,  
(QualifierSourceColumn as string, QualifierSourceValue as string)*) as string
```

Example:

```
dvm:lookupValue ('cityMap.dvm', 'CityCodes', 'BO', 'CityNames',  
'CouldNotBeFound', 'State', 'Massachusetts')
```

Arguments

- `dvmMetadataURI` - The domain value map URI.
- `SourceColumnName` - The source column name.
- `SourceValue` - The source value (an XPath expression bound to the source document of the XSLT transformation).
- `TargetColumnName` - The target column name.
- `DefaultValue` - If the value is not found, then the default value is returned.
- `QualifierSourceColumn`: The name of the qualifier column.
- `QualifierSourceValue`: The value of the qualifier.

46.4.1.2 `dvm:lookupValue1M`

The `dvm:lookupValue1M` function returns an XML document fragment containing values for multiple target columns of a domain value map, where the value for source column is equal to the source value.

```
dvm:lookupValue1M(dvmMetadataURI as string, SourceColumnName as string,
  SourceValue as string, (TargetColumnName as string)?) as nodeset
```

Arguments

- `dvmMetadataURI` - The domain value map URI.
- `SourceColumnName` - The source column name.
- `SourceValue` - The source value (an XPath expression bound to the source document of the XSLT transformation).
- `TargetColumnName` - The name of the target columns. At least one column name should be specified. The question mark symbol (?) indicates that you can specify multiple target column names.

Example

```
dvm:lookupValue1M ('cityMap.dvm', 'CityCode', 'BO', 'CityName', 'CityNickName')
```

The result is:

```
<CityName>Boston</CityName>
<CityNickName>BeanTown</CityNickName>
```

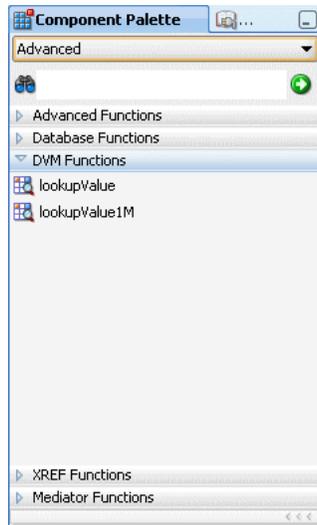
46.4.2 Using Domain Value Map Functions in Transformation

The domain value map functions can be used for transformation with a BPEL service component or a Mediator service component. Transformations are done by using the XSLT Mapper window, which is displayed when you create an XSL file to transform the data from one XML schema to another.

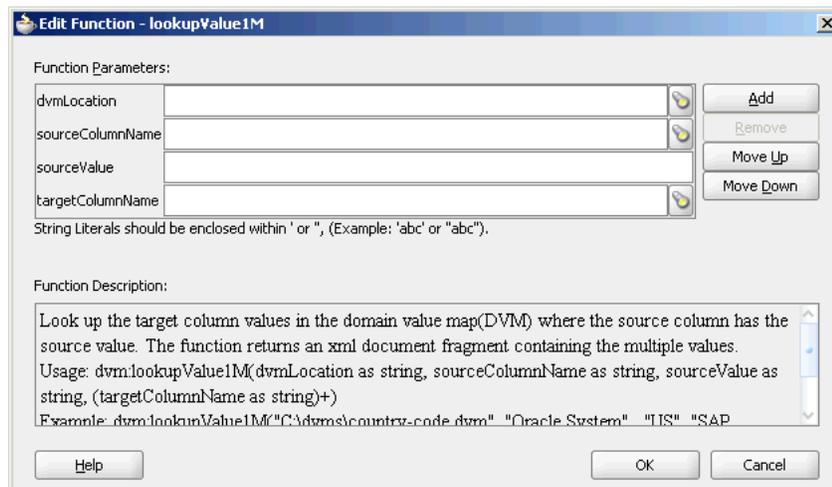
For information about XSLT Mapper, see [Chapter 45, "Creating Transformations with the XSLT Mapper"](#).

To use the lookupValue1M Function in Transformation:

1. In the Application Navigator, double-click an XSL file to open the XSLT Mapper window.
2. In the XSLT Mapper window, expand the trees in the Source and Target panes.
3. In the Component Palette, click the down arrow and then select **Advanced**.
4. Select **DVM Functions** as shown in [Figure 46-5](#).

Figure 46-5 Domain Value Map Functions in Component Palette

5. Drag and drop **lookupValue1M** onto the line that connects the source to the target. A `dvm:lookupValue1M` icon appears on the connecting line.
6. Double-click the **lookupValue1M** icon. The Edit Function – lookupValue1M dialog is displayed, as shown in [Figure 46-6](#).

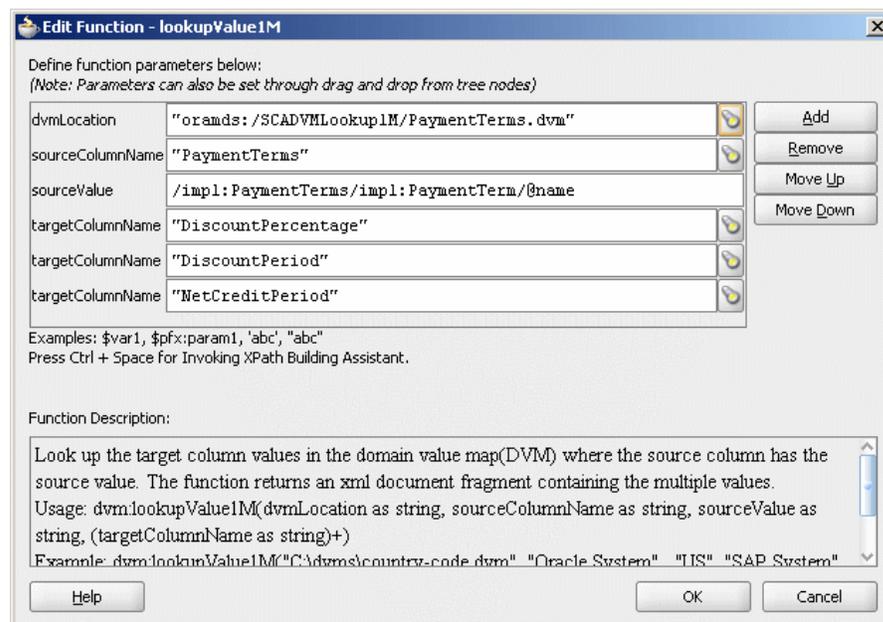
Figure 46-6 Edit Function – lookupValue1M Dialog

7. Specify values for the following fields in the Edit Function – lookupValue1M dialog:

- a. In the **dvmLocation** field, enter the location URI of the domain value map file or click **Browse** to the right of the **dvmLocation** field to select a domain value map file. You can select an already deployed domain value map from MDS and also from the shared location in MDS. This can be done by selecting the Resource Palette.
- b. In the **sourceColumnName** field, enter the name of the domain value map column that is associated with the source element value, or click **Browse** to select a column name from the columns defined for the domain value map you previously selected.
- c. In the **sourceValue** field, enter a value or press **Ctrl-Space** to use XPath Building Assistant. Press the up and down arrow keys to locate an object in the list, and press Enter to select an item.
- d. In the **targetColumnName** field, enter the name of the domain value map column that is associated with the target element value, or click **Browse** to select the name from the columns defined for the domain value map you previously selected.
- e. Click **Add** to add another column as target column and then enter the name of the column.

A populated Edit Function - lookupValue1M dialog is shown in [Figure 46-7](#).

Figure 46-7 Populated Edit Function – lookupValue1M Dialog



8. Click **OK**.

The XSLT Mapper window is displayed with the lookupValue1M function icon.

9. From the **File** menu, click **Save All**.

46.4.3 Using Domain Value Map Functions in XPath Expressions

You can use the domain value map functions to create XPath expressions in the Expression Builder dialog. You can access the Expression builder dialog through the Filter Expressions or the Assign Values functionality of a Mediator service component.

For information about the Assign Values functionality, see [Section 19.2.2.8, "Assigning Values"](#).

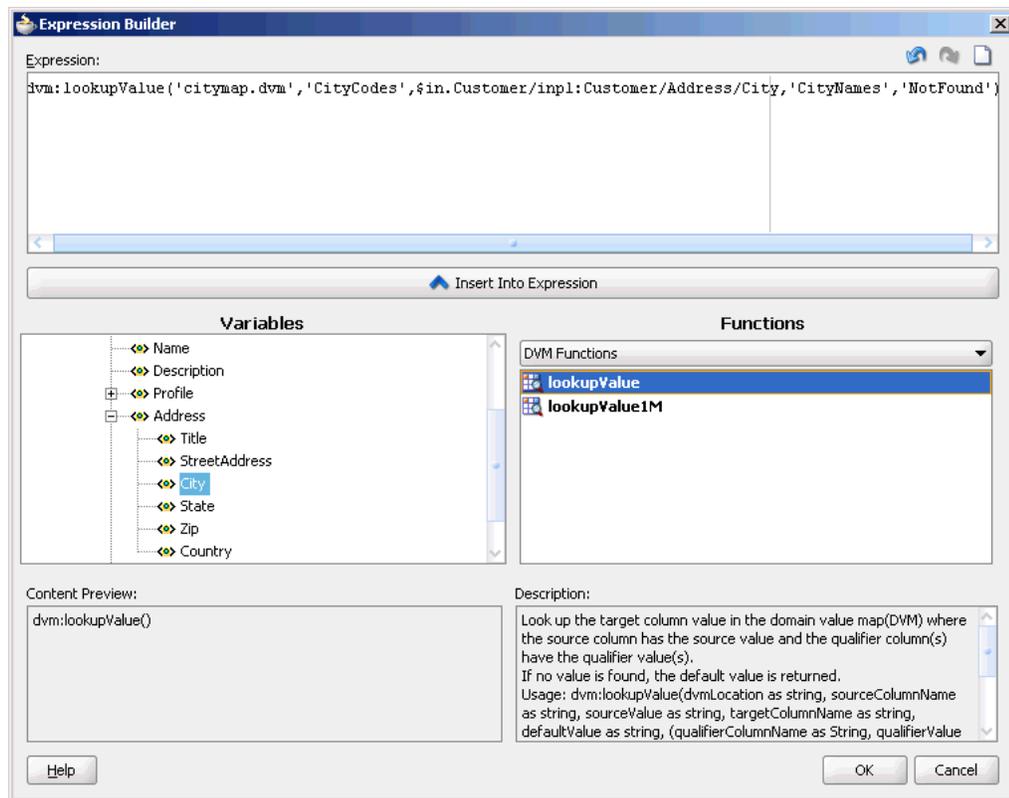
To use the lookupValue function in the Expression Builder dialog:

1. In the Functions list, select **DVM Functions**.
2. Double-click the **dvm:lookupValue** function to add it to the **expression** field.
3. Specify the various arguments of the lookupValue function. For example:

```
dvm:lookupValue('citymap.dvm', 'CityCodes', $in.Customer/inpl:Customer/Address/City, 'CityNames', 'NotFound')
```

This expression, also shown in [Figure 46-8](#), looks up a domain value map for the city name equivalent of a city code. The value of the city code depends on the value specified at runtime.

Figure 46-8 Domain Value Map Functions in the Expression Builder Dialog



46.4.4 What Happens at Runtime

At runtime, a BPEL service component or a Mediator service component uses the domain value map to look up appropriate values.

46.5 Creating a Domain Value Map Use Case for Hierarchical Lookup

This use case demonstrates the hierarchical lookup feature of domain value maps. The hierarchical lookup use case consists of the following steps:

1. Files are picked up from a directory by an adapter service named ReadOrders.

2. The `ReadOrders` adapter service sends the file data to the `ProcessOrders` Mediator.
3. The `ProcessOrders` Mediator then transforms the message to the structure required by the adapter reference. During transformation, Mediator looks up the `UnitsOfMeasure` domain value map for an equivalent value of the `Common` domain.
4. The `ProcessOrders` Mediator sends the message to an external reference `WriteOrders`.
5. The `WriteOrders` reference writes the message to a specified output directory.

For downloading the sample files mentioned in this section, visit the following URL:

http://www.oracle.com/technology/sample_code/products/mediator

46.5.1 Creating the HierarchicalValue Use Case

This section provides the design-time tasks for creating, building, and deploying your SOA composite application. These tasks must be performed in the order in which they are presented.

- [Section 46.5.1.1, "Task 1: Creating an Oracle JDeveloper Application and Project"](#)
- [Section 46.5.1.2, "Task 2: Creating a Domain Value Map"](#)
- [Section 46.5.1.3, "Task 3: Creating a File Adapter Service"](#)
- [Section 46.5.1.4, "Task 4: Creating ProcessOrders Mediator Component"](#)
- [Section 46.5.1.5, "Task 5: Creating a File Adapter Reference"](#)
- [Section 46.5.1.6, "Task 6: Specifying Routing Rules"](#)
- [Section 46.5.1.7, "Task 7: Configuring Oracle Application Server Connection"](#)
- [Section 46.5.1.8, "Task 8: Deploying the Composite Application"](#)

46.5.1.1 Task 1: Creating an Oracle JDeveloper Application and Project

To create an application and a project for the use case:

1. In Oracle JDeveloper, click **File** and select **New**.
The New Gallery dialog appears.
2. In the New Gallery, expand the **General** node, and select the **Applications** category.
3. In the **Items** list, select **SOA Application** and click **OK**.
The Create SOA Application Wizard appears.
4. In the **Application Name** field, enter `Hierarchical` and then click **Next**.
The **Name your SOA project** screen appears.
5. In the **Project Name** field, enter `HierarchicalValue` and click **Next**.
The **Configure SOA settings for the SOA project** screen appears.
6. In the Composite Template list, select **Empty Composite** and then click **Finish**.
The Applications Navigator of Oracle JDeveloper is populated with the new application and the project, and the Design tab contains a blank palette.
7. From the **File** menu, click **Save All**.

46.5.1.2 Task 2: Creating a Domain Value Map

After creating an application and a project for the use case, you must create a domain value map.

To create a domain value map:

1. In the Application Navigator, right-click the **HierarchicalValue** project and select **New**.
2. In the New Gallery dialog, expand the **SOA Tier** node, and then select the **Transformations** category.
3. In the Items list, select **Domain Value Map(DVM)** and click **OK**.

The Create Domain Value Map(DVM) File dialog is displayed.

4. In the **File Name** field, enter `UnitsOfMeasure.dvm`.
5. In the **Domain Name** fields, enter `Siebel` and `Common`.
6. In the **Domain Value** field corresponding to the Siebel domain, enter `Ea`.
7. In the **Domain Value** field corresponding to the Common domain, enter `Each`.
8. Click **OK**.

The Domain Value Map Editor is displayed.

9. Click **Add** and then select **Add Column**.

The Create DVM Column dialog is displayed.

10. In the **Name** field, enter `TradingPartner`.
11. In the **Qualifier** list, select **true**.
12. In the **QualifierOrder** field, enter `1` and click **OK**.
13. Repeat Step 9 through Step 12 to create another qualifier named `StandardCode` with qualifier order as `2`.

14. Click **Add** and then select **Add Row**.

Repeat this step to add two more rows.

15. Enter the following information in the newly added rows of the domain value map table:

Siebel	Common	TradingPartner	StandardCode
EC	Each		OAG
E-RN	Each	A.C.Networks	RN
EO	Each	ABC Inc	RN

The Domain Value Map Editor would appear as shown in [Figure 46-9](#).

Figure 46–9 UnitsOfMeasure Domain Value Map

The screenshot shows the 'Domain Value Map(DVM)' editor. The 'Name' field is 'UnitsOfMeasure'. The 'Description' field is empty. Below is a 'Map Table' with the following data:

Siebel	Common	TradingPartner	StandardCode
Ea	Each		
Ec	Each		OAG
E-RN	Each	A, C, Networks	RN
EO	Each	ABC Inc	RN

16. From the **File** menu, click **Save All** and close the Domain Value Map Editor.

46.5.1.3 Task 3: Creating a File Adapter Service

After creating the domain value map, you must create a File adapter service, named `ReadOrders` to read the XML files from a directory.

Note: Oracle Mediator may process the same file twice when run against Oracle RAC planned outages. This is because a File adapter is a non-XA compliant adapter. So, when it participates in a global transaction, it may not follow the XA interface specification of processing each file once and only once.

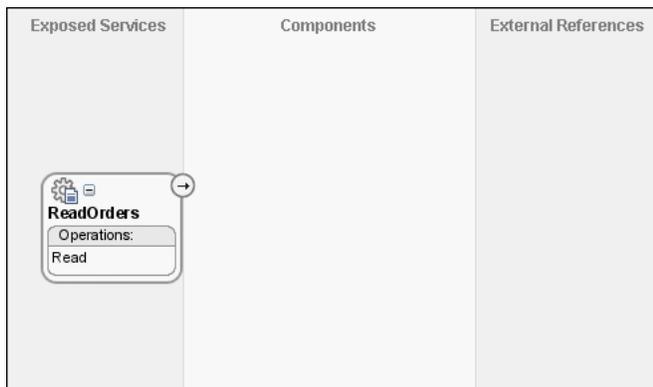
To create a File adapter service:

1. From the Components Palette, select **SOA**.
2. Select **File Adapter** and drag it to the Exposed Services design area.
3. If the Adapter Configuration Wizard Welcome page appears, click **Next**.
The Service Name page is displayed.
4. In the **Service Name** field, enter `ReadOrders` and then click **Next**.
The Operation page is displayed.
5. In the **Operation Type** field, select **Read File** and then click **Next**.
The File Directories page is displayed.
6. In the **Directory for Incoming Files (physical path)** field, enter the directory from which you want to read the files.
7. Click **Next**.
The File Filtering page is displayed.
8. In the **Include Files with Name Pattern** field, enter `*.xml` and then click **Next**.

- The File Polling page is displayed.
9. Change the **Polling Frequency** field value to **10 seconds** and then click **Next**.
The Messages page is displayed.
 10. Click **Search**.
The Type Chooser dialog is displayed.
 11. Click **Import Schema File**.
The Import Schema File dialog is displayed.
 12. Click **Search** and select the **Order.xsd** file present in the `Samples` folder.
 13. Click **OK**.
 14. Expand the navigation tree to **Type Explorer\Imported Schemas\Order.xsd**.
 15. Select **listOfOrder** and click **OK**.
 16. Click **Next**.
The Finish page is displayed.
 17. Click **Finish**.
 18. From the **File** menu, click **Save All**.

Figure 46–10 shows the `ReadOrders` service in SOA Composite Editor.

Figure 46–10 ReadOrders Service in the SOA Composite Editor



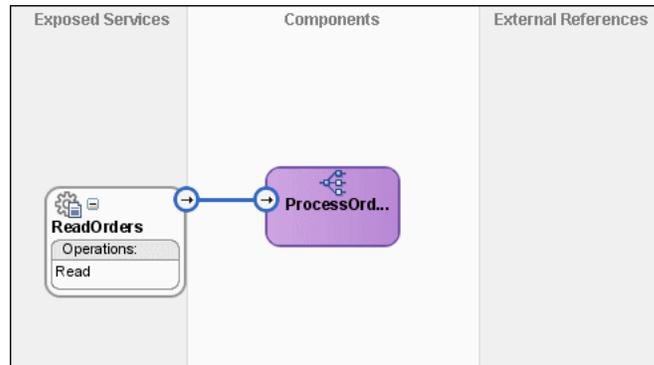
46.5.1.4 Task 4: Creating ProcessOrders Mediator Component

To create a Mediator component named ProcessOrders:

1. Drag and drop a Mediator from Component Palette to the Components design area.
The Create Mediator dialog is displayed.
2. In the **Name** field, enter `ProcessOrders`.
3. In the **Template** list, select **Define Interface Later**.
4. Click **OK**.
A Mediator with name `ProcessOrders` is created.
5. In the SOA Composite Editor, connect the `ReadOrders` service to the `ProcessOrders` Mediator, as shown in Figure 46–11.

This specifies the file adapter service to invoke the `ProcessOrders` Mediator while reading a file from the input directory.

Figure 46–11 *ReadOrders Service Connected to the ProcessOrders Mediator*



6. From the **File** menu, click **Save All**.

46.5.1.5 Task 5: Creating a File Adapter Reference

To create a file adapter reference:

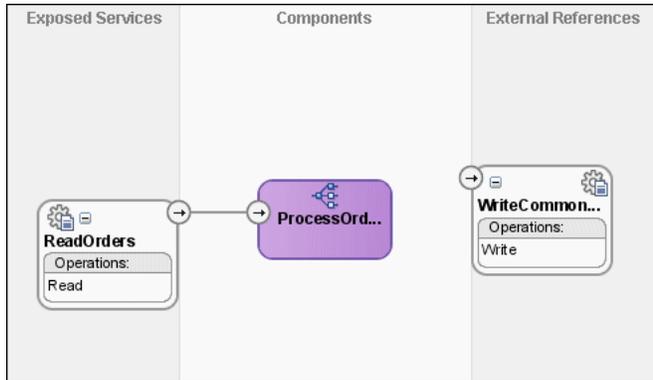
1. From the Components Palette, select **SOA**.
2. Select **File Adapter** and drag it to the External References design area.
The Adapter Configuration wizard Welcome page is displayed.
3. Click **Next**.
The Service Name page is displayed.
4. In the **Service Name** field, enter `WriteCommonOrder`.
5. Click **Next**.
The Operation page is displayed.
6. In the **Operation Type** field, select **Write File**.
7. Click **Next**.
The File Configuration page is displayed.
8. In the **Directory for Outgoing Files (physical path)** field, enter the name of the directory where you want to write the files.
9. In the **File Naming Convention** field, enter `common_order_%SEQ%.xml` and click **Next**.
The Messages page is displayed.
10. Click **Search**.
The Type Chooser dialog is displayed.
11. Navigate to **Type Explorer, Project Schema Files, Order.xsd** and then select **listOfOrder**.
12. Click **OK**.
13. Click **Next**.

The Finish page is displayed.

14. Click **Finish**.

Figure 46–12 shows the `WriteCommonOrder` reference in SOA Composite Editor.

Figure 46–12 WriteCommonOrder Reference in SOA Composite Editor



15. From the **File** menu, click **Save All**.

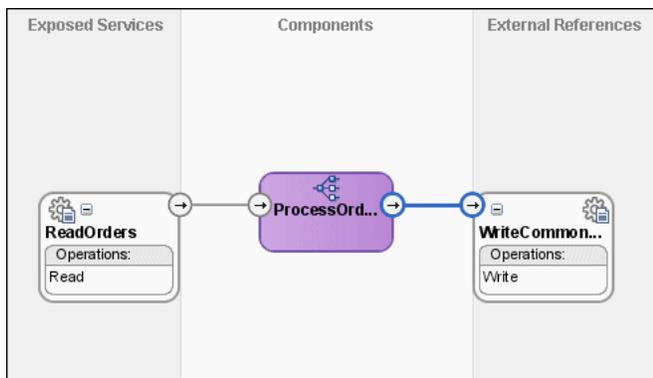
46.5.1.6 Task 6: Specifying Routing Rules

You must specify the path that messages take from the `ReadOrders` adapter service to the external reference.

To specify routing rules

1. Connect the `ProcessOrders` Mediator to the `WriteCommonOrder` reference as shown in Figure 46–13.

Figure 46–13 ProcessOrders Mediator Connected to the WriteCommonOrder Reference



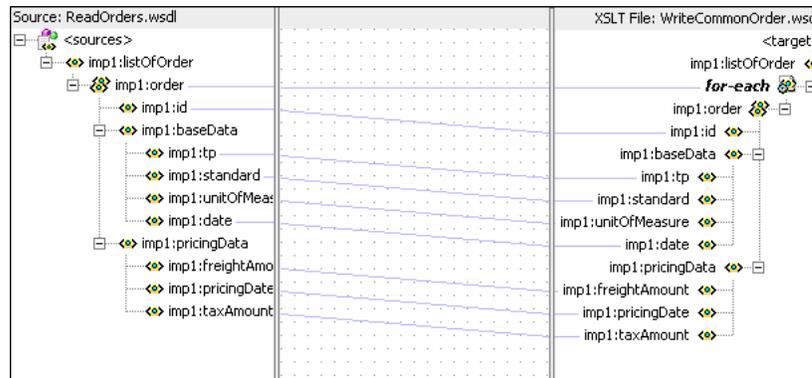
2. Double-click `ProcessOrders` Mediator.
3. Click the icon to the right of the **Transform Using** field.
The Request Transformation Map dialog is displayed.
4. Select **Create New Mapper File** and click **OK**.
A `listOfOrder_To_listOfOrder.xsl` tab is displayed.
5. Drag and drop the `imp1:listOfOrder` source element onto `imp1:listOfOrder` target element.

The Auto Map Preferences dialog is displayed.

6. From the **During Auto Map** options, deselect **Match Elements Considering their Ancestor Names**.
7. Click **OK**.

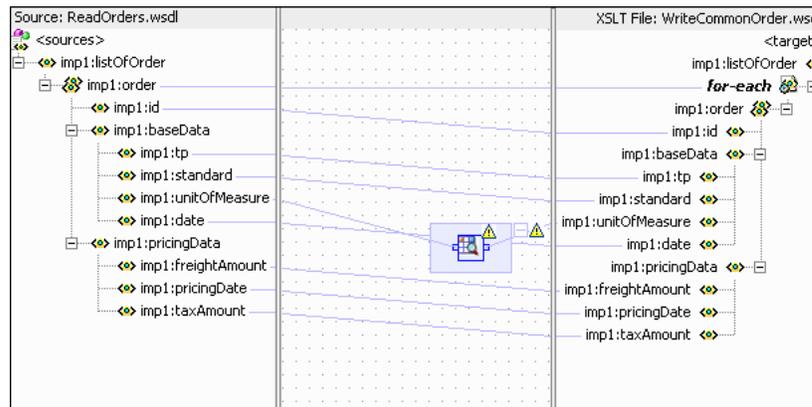
The `listOfOrder_To_listOfOrder.xsl` tab appears as shown in [Figure 46-14](#).

Figure 46-14 *imp1:listOfOrder To imp1:listOfOrder Transformation*



8. In the Components Palette, select **Advanced**.
9. Click **DVM Functions**.
10. Drag and drop **lookupValue** on the line connecting the `unitsOfMeasure` elements, as shown in [Figure 46-15](#).

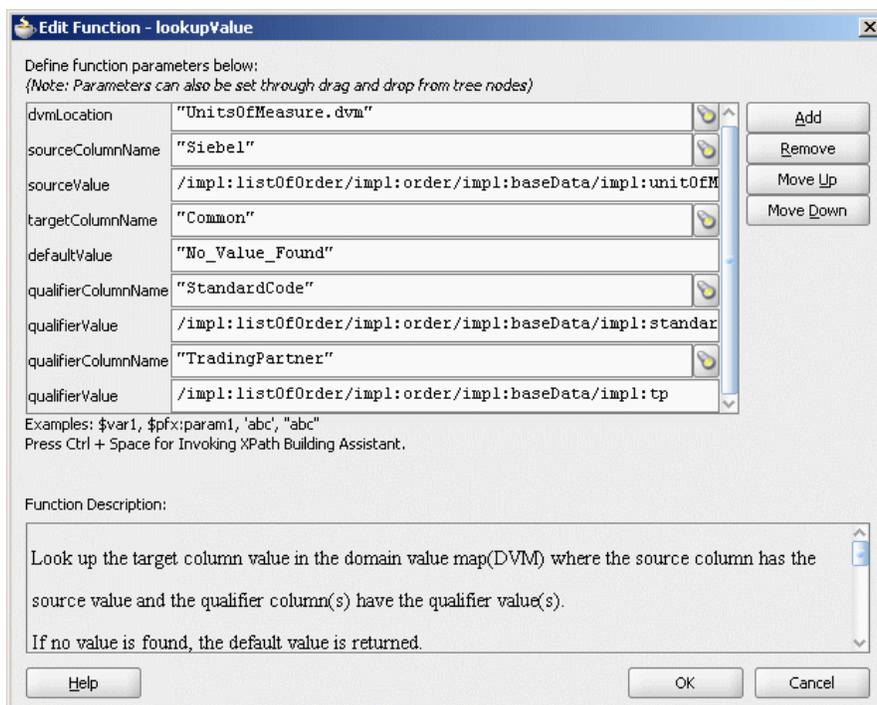
Figure 46-15 *Adding lookupValue Function to imp1:listOfOrder To imp1:listOfOrder.xsl*



11. Double-click the **lookupvalue** icon.
The Edit Function-lookupValue dialog is displayed.
12. Click **Search** to the right of the **dvmLocation** field.
The SCA Resource Lookup dialog is displayed.
13. Select **UnitsofMeasure.dvm** and click **OK**.
14. Click **Search** to the right of the **sourceColumnName** field.
The Select DVM Column dialog is displayed.

15. Select **Siebel** and click **OK**.
16. In the **sourceValue** column, enter
`/impl:listOfOrder/impl:order/impl:baseData/impl:unitOfMeasure.`
17. Click **Search** to the right of the **targetColumnName** field.
 The Select DVM Column dialog is displayed.
18. Select **Common** and click **OK**.
19. In the **defaultValue** field, enter "No_Value_Found".
20. Click **Add**.
 A **qualifierColumnName** row is added.
21. In the **qualifierColumnName** field, enter "StandardCode".
22. Click **Add**.
 A **qualifierValue** row is added.
23. In the **qualifierValue** field, enter
`/impl:listOfOrder/impl:order/impl:baseData/impl:standard.`
24. Click **Add** to insert another **qualifierColumnName** row.
25. In the **qualifierColumnName** field, enter "TradingPartner".
26. Click **Add** to insert another **qualifierValue** row.
27. In the **qualifierValue** field, enter
`/impl:listOfOrder/impl:order/impl:baseData/impl:tp.`
 The Edit Function-lookupValue dialog would appear as shown in [Figure 46–16](#).

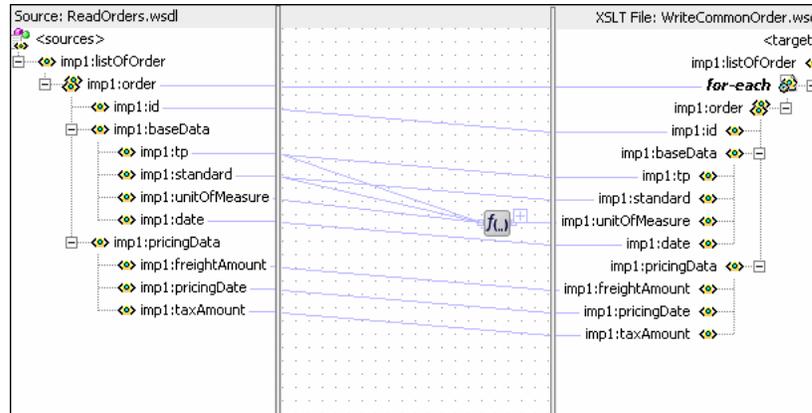
Figure 46–16 Edit Function-lookupValue Function Dialog: Hierarchical Lookup Use Case



28. Click OK.

The transformation would appear as shown in [Figure 46–17](#).

Figure 46–17 Complete `imp1:listOfOrder` To `imp1:listOfOrder` Transformation



29. From the **File** menu, click **Save All** and close the `listOfOrder_To_listOfOrder.xml` tab.

46.5.1.7 Task 7: Configuring Oracle Application Server Connection

An Oracle Application Server connection is required for deploying your SOA composite application. For information on creating an Oracle Application Server connection, refer to *Oracle Fusion Middleware User's Guide for Technology Adapters*.

46.5.1.8 Task 8: Deploying the Composite Application

Deploying the `HierarchicalValue` composite application to Oracle Application Server consists of the following steps:

- Creating an Application Deployment Profile.
- Deploying the Application to Oracle Application Server.

For detailed information about these steps, see [Section 43.2, "Deploying a Single SOA Composite in Oracle JDeveloper"](#).

46.5.2 Running and Monitoring the HierarchicalValue Application

After deploying the `HierarchicalValue` application, you can run it by copying the input XML file `sampleorder.xml` to the input folder. This file is available in the `samples` folder. On successful completion, a file with name `common_order_1.xml` is written to the specified output directory.

For monitoring the running instance, you can use Oracle Enterprise Manager Console at the following URL:

```
http://hostname:port/em
```

where `hostname` is the host on which you installed the Oracle SOA Suite infrastructure.

For detailed information about these steps, see [Section 43.2, "Deploying a Single SOA Composite in Oracle JDeveloper"](#).

46.6 Creating a Domain Value Map Use Case For Multiple Values

This use case demonstrates the integration scenario using a DVM lookup between two endpoints to look up multiple values. For example, if the inbound is State, then the outbound are Shortname of State, Language, and Capital. The multivalued lookup use case consists of the following steps:

1. Files are picked up from a directory by an adapter service named `readFile`.
2. The `readFile` adapter service sends the file data to the `LookupMultipleValuesMediator` Mediator.
3. The `LookupMultipleValuesMediator` Mediator then transforms the message to the structure required by the adapter reference. During transformation, Mediator looks up the multivalued domain value map for an equivalent value of Longname and Shortname domains.
4. The `LookupMultipleValuesMediator` Mediator sends the message to an external reference `writeFile`.
5. The `writeFile` reference writes the message to a specified output directory.

For downloading the sample files mentioned in this section, visit the following URL:

http://www.oracle.com/technology/sample_code/products/mediator

46.6.1 Creating the Multivalued Use Case

This section provides the design-time tasks for creating, building, and deploying your SOA composite application. These tasks should be performed in the order in which they are presented.

- [Section 46.6.1.1, "Task 1: Creating an Oracle JDeveloper Application and Project"](#)
- [Section 46.6.1.2, "Task 2: Creating a Domain Value Map"](#)
- [Section 46.6.1.3, "Task 3: Creating a File Adapter Service"](#)
- [Section 46.6.1.4, "Task 4: Creating LookupMultipleValuesMediator Mediator Component"](#)
- [Section 46.6.1.5, "Task 5: Creating a File Adapter Reference"](#)
- [Section 46.6.1.6, "Task 6: Specifying Routing Rules"](#)
- [Section 46.6.1.7, "Task 7: Configuring Oracle Application Server Connection"](#)
- [Section 46.6.1.8, "Task 8: Deploying the Composite Application"](#)

46.6.1.1 Task 1: Creating an Oracle JDeveloper Application and Project

To create an application and a project for the use case:

1. In Oracle JDeveloper, click **File** and select **New**.
The New Gallery dialog appears.
2. In the New Gallery, expand the **General** node, and select the **Applications** category.
3. In the **Items** list, select **SOA Application** and click **OK**.
The Create SOA Application Wizard appears.
4. In the **Application Name** field, enter `Multivalued` and then click **Next**.

The Name your project screen appears.

5. In the **Project Name** field, enter `Multivalued` and click **Next**.

The Configure SOA settings screen appears.

6. In the Composite Template list, select **Empty Composite** and then click **Finish**.

The Applications Navigator of Oracle JDeveloper is populated with the new application and the project, and the Design tab contains a blank palette.

7. From the **File** menu, click **Save All**.

46.6.1.2 Task 2: Creating a Domain Value Map

After creating an application and a project for the use case, you must create a domain value map.

To create a domain value map:

1. In the Application Navigator, right-click the **Multivalued** project and select **New**.
2. In the New Gallery dialog, expand the **SOA Tier** node, and then select the **Transformations** category.
3. In the Items list, select **Domain Value Map(DVM)** and click **OK**.

The Create Domain Value Map(DVM) File dialog is displayed.

4. In the **File Name** field, enter `multivalued.dvm`.
5. In the **Domain Name** fields, enter `Longname`, `Shortname`, `Language`, and `Capital`.
6. In the **Domain Value** field corresponding to the `Longname` domain, enter `Karnataka`.
7. In the **Domain Value** field corresponding to the `Shortname` domain, enter `KA`.
8. In the **Domain Value** field corresponding to the `Language` domain, enter `Kannada`.
9. In the **Domain Value** field corresponding to the `Capital` domain, enter `Bangalore`.
10. Click **OK**.

The Domain Value Map Editor is displayed.

11. Click **Add** and then select **Add Row**.

Repeat this step to add two more rows.

12. Enter the following information in the newly added rows of the domain value map table:

Longname	Shortname	Language	Capital
Karnataka	KA	Kannada	Bangalore
Tamilnadu	TN	Tamil	Chennai
Andhrapradesh	AP	Telugu	Hyderabad
Kerala	KL	Malayalam	Trivandram

The Domain Value Map Editor would appear as shown in [Figure 46–18](#).

Figure 46–18 Multivalue Domain Value Map

 **Domain Value Map(DVM)**

Name:

Description:

Map Table:   

Longname	Shortname	Language	Capital
Karnataka	KA	Kannada	Bangalore
Tamilnadu	TN	Tamil	Chennai
Andhrapradesh	AP	Telugu	Hyderabad
Kerala	KL	Malayalam	Trivandrum

13. From the **File** menu, click **Save All** and close the Domain Value Map Editor.

46.6.1.3 Task 3: Creating a File Adapter Service

After creating the domain value map, you must create a File adapter service, named `readFile` to read the XML files from a directory.

Note: Oracle Mediator may process the same file twice when run against Oracle RAC planned outages. This is because a File adapter is a non-XA compliant adapter. So, when it participates in a global transaction, it may not follow the XA interface specification of processing each file once and only once.

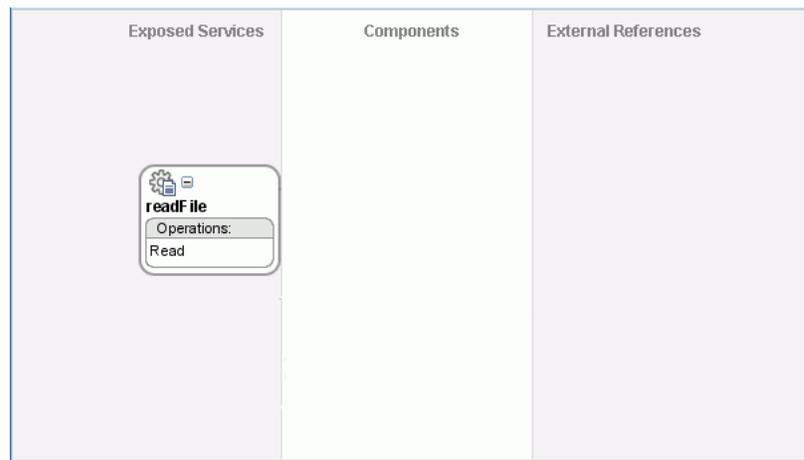
To create a File adapter service:

- From the Components Palette, select **SOA**.
- Select **File Adapter** and drag it to the Exposed Services design area.
- If the Adapter Configuration Wizard Welcome page appears, click **Next**.
The Service Name page is displayed.
- In the **Service Name** field, enter `readFile` and then click **Next**.
The Adapter Interface page is displayed.
- Click **Define from operation and schema (specified later)** and then click **Next**.
The Operation page is displayed.
- In the **Operation Type** field, select **Read File** and then click **Next**.
The File Directories page is displayed.
- In the **Directory for Incoming Files (physical path)** field, enter the directory from which you want to read the files.
- Click **Next**.
The File Filtering page is displayed.
- In the **Include Files with Name Pattern** field, enter `*.xml` and then click **Next**.
The File Polling page is displayed.

10. Change the **Polling Frequency** field value to **1 second** and then click **Next**.
The Messages page is displayed.
11. Click **Search**.
The Type Chooser dialog is displayed.
12. Click **Import Schema File**.
The Import Schema File dialog is displayed.
13. Click **Search** and select the **input.xsd** file present in the `Samples` folder.
14. Click **OK**.
15. Expand the navigation tree to **Type Explorer\Imported Schemas\input.xsd**.
16. Select **Root-Element** and click **OK**.
17. Click **Next**.
The Finish page is displayed.
18. Click **Finish**.
19. From the **File** menu, click **Save All**.

Figure 46–19 shows the `readFile` service in SOA Composite Editor.

Figure 46–19 *readFile Service in the SOA Composite Editor*



46.6.1.4 Task 4: Creating LookupMultiplevaluesMediator Mediator Component

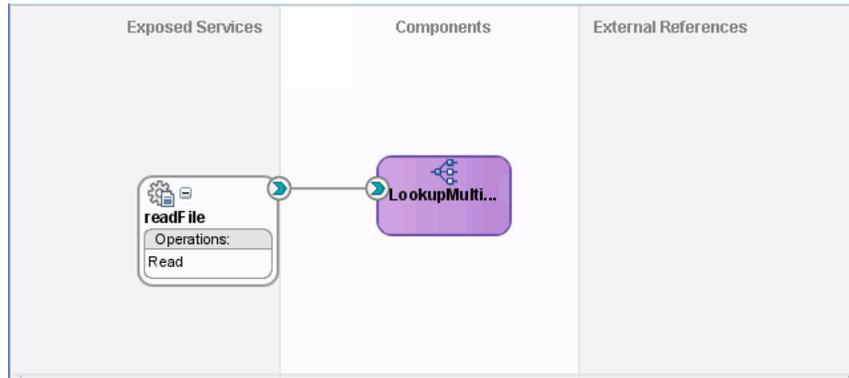
To create a Mediator component named LookupMultiplevaluesMediator:

1. Drag and drop a Mediator from Component Palette to the Components design area.
The Create Mediator dialog is displayed.
2. In the **Name** field, enter `LookupMultiplevaluesMediator`.
3. In the **Template** list, select **Define Interface Later**.
4. Click **OK**.
A Mediator with name `LookupMultiplevaluesMediator` is created.

5. In the SOA Composite Editor, connect the `readFile` service to the `LookupMultipleValuesMediator` Mediator, as shown in [Figure 46–20](#).

This specifies the file adapter service to invoke the `LookupMultipleValuesMediator` Mediator while reading a file from the input directory.

Figure 46–20 *readFile Service Connected to the LookupMultipleValuesMediator Mediator*



6. From the **File** menu, click **Save All**.

46.6.1.5 Task 5: Creating a File Adapter Reference

To create a file adapter reference:

1. From the Components Palette, select **SOA**.
2. Select **File Adapter** and drag it to the External References design area.
The Adapter Configuration wizard Welcome page is displayed.
3. Click **Next**.
The Service Name page is displayed.
4. In the **Service Name** field, enter `writeFile` and then click **Next**.
The Adapter Interface page is displayed.
5. Click **Define from operation and schema (specified later)** and then click **Next**.
The Operation page is displayed.
6. Click **Next**.
The Operation page is displayed.
7. In the **Operation Type** field, select **Write File**.
8. Click **Next**.
The File Configuration page is displayed.
9. In the **Directory for Outgoing Files (physical path)** field, enter the name of the directory where you want to write the files.
10. In the **File Naming Convention** field, enter `multivalue_%SEQ%.xml` and click **Next**.
The Messages page is displayed.
11. Click **Search**.

The Type Chooser dialog is displayed.

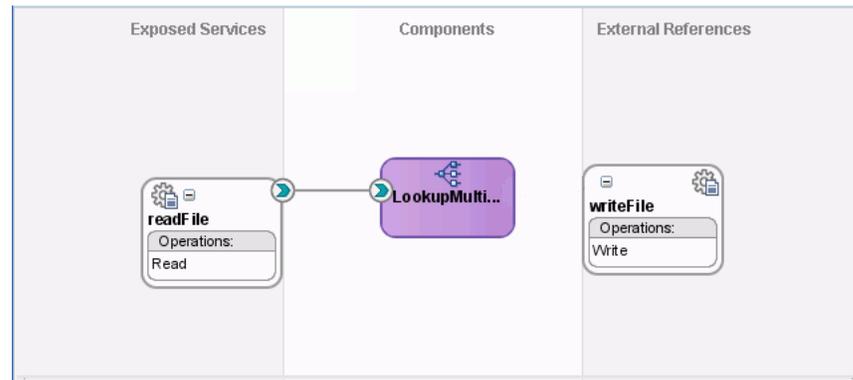
12. Navigate to **Type Explorer, Project Schema Files, output.xsd** and then select **Root-Element**.
13. Click **OK**.
14. Click **Next**.

The Finish page is displayed.

15. Click **Finish**.

Figure 46–21 shows the `writeFile` reference in SOA Composite Editor.

Figure 46–21 *writeFile Reference in SOA Composite Editor*



16. From the **File** menu, click **Save All**.

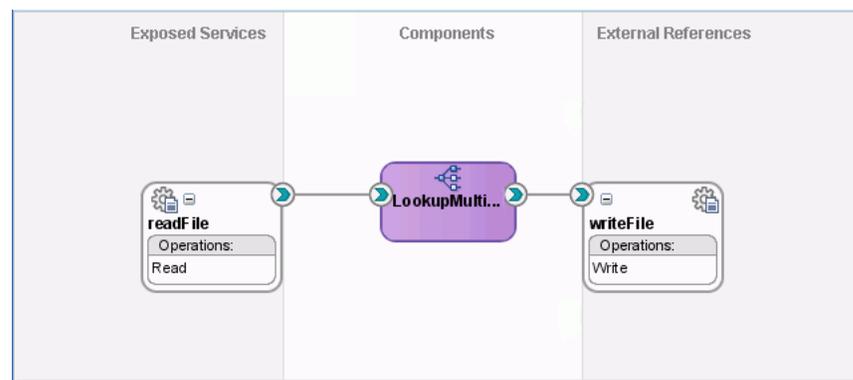
46.6.1.6 Task 6: Specifying Routing Rules

You must specify the path that messages take from the `readFile` adapter service to the external reference.

To specify routing rules

1. Connect the `LookupMultipleValuesMediator` Mediator to the `writeFile` reference as shown in Figure 46–22.

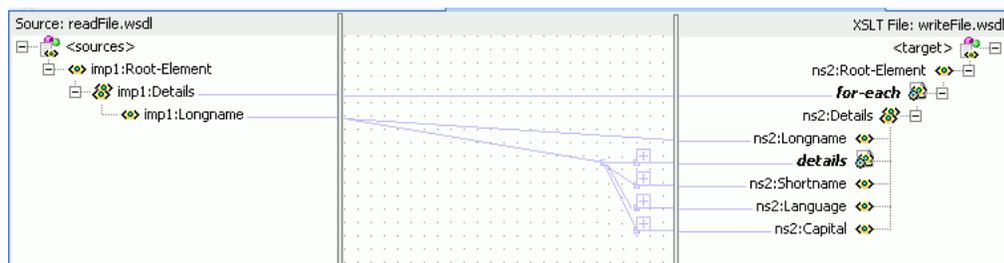
Figure 46–22 *LookupMultipleValuesMediator Mediator Connected to the writeFile Reference*



2. Double-click the `LookupMultipleValuesMediator` Mediator.

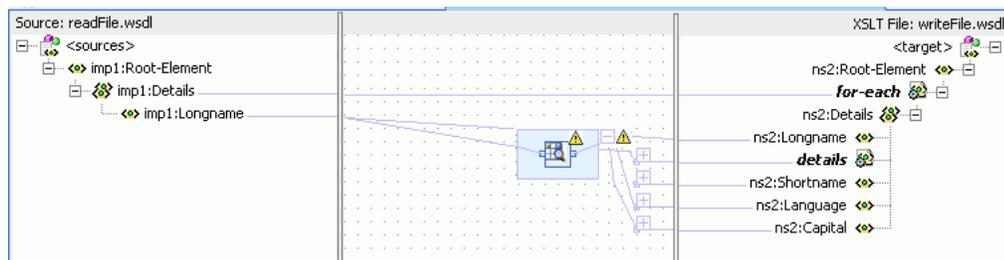
3. Click the icon to the right of the **Transform Using** field.
The Request Transformation Map dialog is displayed.
4. Select **Create New Mapper File** and click **OK**.
A `Input_To_Output_with_multiple_values_lookup.xsl` tab is displayed.
5. Drag and drop the `imp1:Root-Element` source element to `ns2:Root-Element` target element.
The Auto Map Preferences dialog is displayed.
6. From the **During Auto Map** options, deselect **Match Elements Considering their Ancestor Names**.
7. Click **OK**.
The `Input_To_Output_with_multiple_values_lookup.xsl` tab appears as shown in [Figure 46-23](#).

Figure 46-23 `imp1:Root-Element` To `ns2:Root-Element` Transformation



8. In the Components Palette, select **Advanced**.
9. Click **DVM Functions**.
10. Drag and drop `lookupValue1M` to the center swim lane, as shown in [Figure 46-24](#).

Figure 46-24 Adding `lookupValue` Function to `imp1:Root-Element` to `ns2:Root-Element`

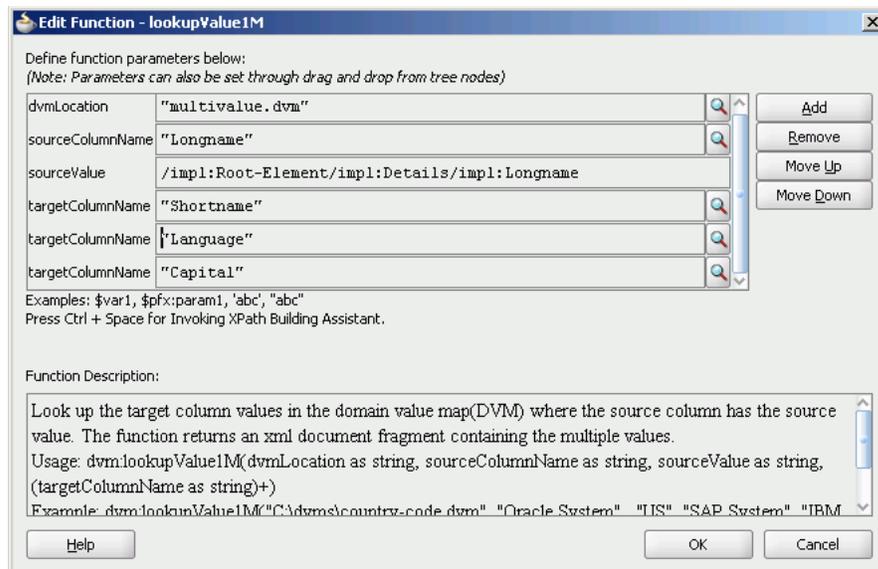


11. Double-click the `lookupvalue1M` icon.
The Edit Function-lookupValue1M dialog is displayed.
12. Click **Search** to the right of `dvmLocation` field.
The SCA Resource Lookup dialog is displayed.
13. Select `multivalue.dvm` and click **OK**.
14. Click **Search** to the right of `sourceColumnName` field.
The Select DVM Column dialog is displayed.

15. Select **Longname** and click **OK**.
16. In the **sourceValue** column, enter
/impl:Root-Element/impl:Details/impl:Longname.
17. Click **Search** to the right of **targetColumnName** field.
The Select DVM Column dialog is displayed.
18. Select **Shortname** and click **OK**.
19. Click **Add**.
A targetColumnName row is added.
20. In the **targetColumnName** field, enter "Language".
21. Click **Add** to insert another targetColumnName row.
22. In the **targetColumnName** field, enter "Capital".

The Edit Function-lookupValue dialog would appear as shown in [Figure 46–25](#).

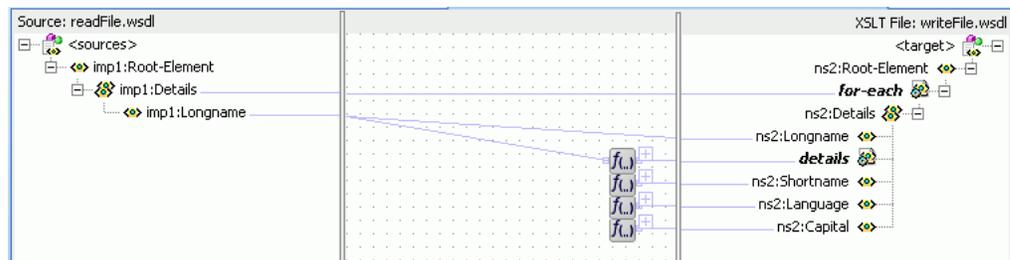
Figure 46–25 Edit Function-lookupValue Function Dialog: Multiple Value Lookup Use Case



23. Click **OK**.

The Transformation would appear as shown in [Figure 46–26](#).

Figure 46–26 Complete impl:Root-Element To ns2:Root-Element Transformation



24. From the **File** menu, click **Save All** and close the `Input_To_Output_with_multiple_values_lookup.xsl` tab.

46.6.1.7 Task 7: Configuring Oracle Application Server Connection

An Oracle Application Server connection is required for deploying your SOA composite application. For information on creating Oracle Application Server connection, refer to *Oracle Fusion Middleware User's Guide for Technology Adapters*.

46.6.1.8 Task 8: Deploying the Composite Application

Deploying the `Multivalued` composite application to Oracle Application Server consists of the following steps:

- Creating an Application Deployment Profile.
- Deploying the application to Oracle Application Server.

For detailed information about these steps, see [Section 43.2, "Deploying a Single SOA Composite in Oracle JDeveloper"](#).

46.6.2 Running and Monitoring the Multivalued Application

After deploying the `Multivalued` application, you can run it by copying the input XML file `sampleinput.xml` to the input folder. This file is available in the `samples` folder. On successful completion, a file with name `multivalued_1.xml` is written to the specified output directory.

For monitoring the running instance, you can use Oracle Enterprise Manager Console at the following URL:

```
http://hostname:port/em
```

where `hostname` is the host on which you installed the Oracle SOA Suite infrastructure.

In Oracle Enterprise Manager Console, you can click the `Multivalued` to see the project dashboard.

To view the detailed execution trail, click the instance ID in the instance column. The Flow Trace page is displayed.

Working with Cross References

This chapter describes how to use cross referencing feature of Oracle SOA Suite to associate identifiers for equivalent entities created in different applications.

This chapter includes the following sections:

- [Section 47.1, "Introduction to Cross References"](#)
- [Section 47.2, "Creating and Modifying Cross Reference Tables"](#)
- [Section 47.3, "Populating Cross Reference Tables"](#)
- [Section 47.4, "Looking Up Cross Reference Tables"](#)
- [Section 47.5, "Deleting a Cross Reference Table Value"](#)
- [Section 47.6, "Creating and Running Cross Reference Use Case"](#)
- [Section 47.7, "Creating and Running Cross Reference for 1M Functions"](#)

47.1 Introduction to Cross References

Cross references enable you to dynamically map values for equivalent entities created in different applications.

Note: The Cross Referencing feature enables you to dynamically integrate values between applications, whereas the domain value maps enables you to specify values at design time. For more information about domain value maps, see [Chapter 46, "Working with Domain Value Maps"](#).

When you create or update objects in one application, you may also want to propagate the changes to other application. For example, when a new customer is created in a SAP application, you might want to create a new entry for the same customer in your Oracle E-Business Suite application named as EBS. However, the applications that you are integrating could be using different entities to represent the same information. For example, for each new customer in a SAP application, a new row is inserted in its Customer database with a unique identifier such as SAP_001. When the same information is propagated to an Oracle E-Business Suite application and a Siebel application, the new row should be inserted with different identifiers such as EBS_1001 and SBL001. In such cases, you need some kind of functionality to map these identifiers with each other so that they could be interpreted by different applications to be referring to the same entity. This can be done by using the cross references.

Cross references are stored in the form of tables. [Table 47-1](#) shows a cross reference table containing information about customer identifiers in different applications.

Table 47-1 Cross Reference Table Sample

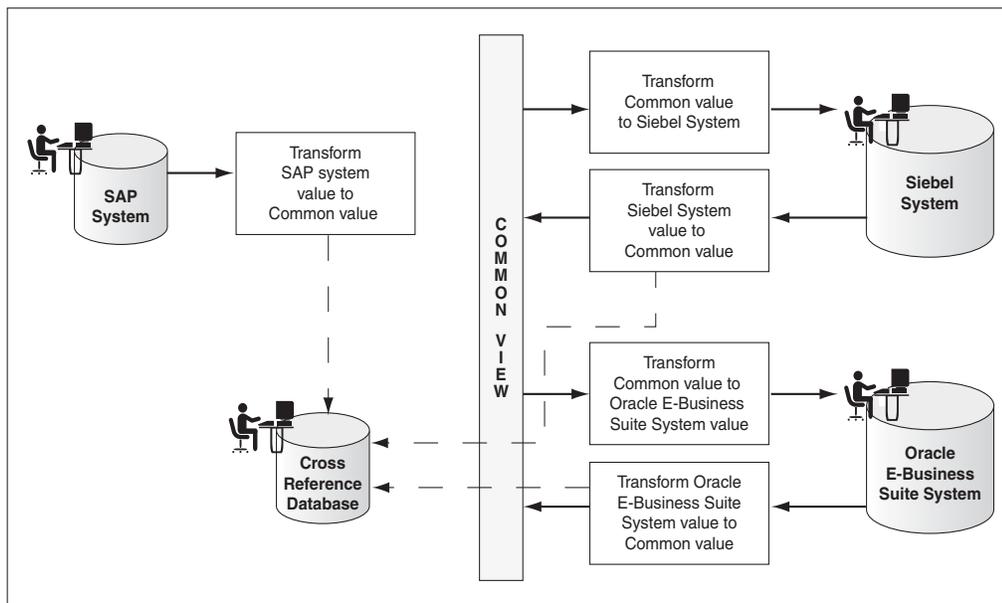
SAP	EBS	SBL
SAP_001	EBS_1001	SBL001
SAP_002	EBS_1002	SBL002

The identifier mapping is also required when information about a customer is updated in one application and the changes must be propagated in other applications. You can integrate different identifiers by using a common value integration pattern, which maps to all identifiers in a cross reference table. For example, you can add one more column named `Common` to the cross reference table shown in [Table 47-1](#). The updated cross reference table would appear, as shown in [Table 47-2](#).

Table 47-2 Cross Reference Table with Common Column

SAP	EBS	SBL	Common
SAP_001	EBS_1001	SBL001	CM001
SAP_002	EBS_1002	SBL002	CM002

[Figure 47-1](#) shows how you can use common value integration pattern to map identifiers in different applications.

Figure 47-1 Common Value Integration Pattern Example

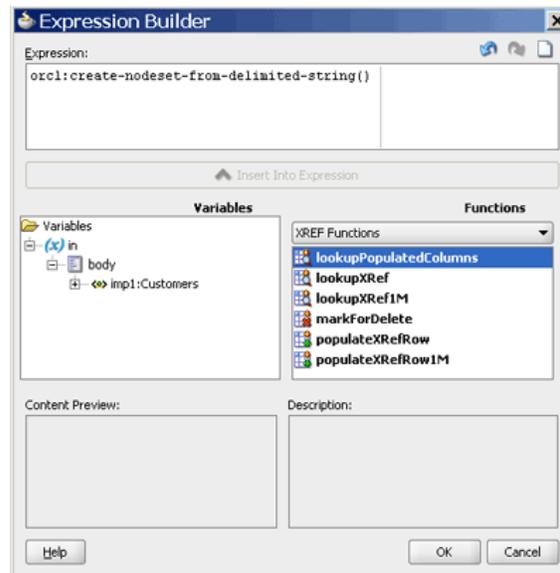
A cross reference table consists of the following two parts: metadata and the actual data. The metadata is the `.xref` file created in Oracle JDeveloper, and is stored in Metadata Services (MDS) as an XML file. By default, the actual data is stored in the `XREF_DATA` table of the database in the SOA infrastructure database schema.

You can create a cross reference table in a SOA composite application of Oracle JDeveloper and then use it to look up for column values at runtime. However, before using a cross reference to lookup a particular value, you must populate it at runtime. This can be done by using the cross reference XPath functions. The XPath functions enable you to populate a cross reference, perform lookups, and delete a column value.

These XPath functions can be used in the Expression builder dialog to create an expression or in the XSLT Mapper to create transformations.

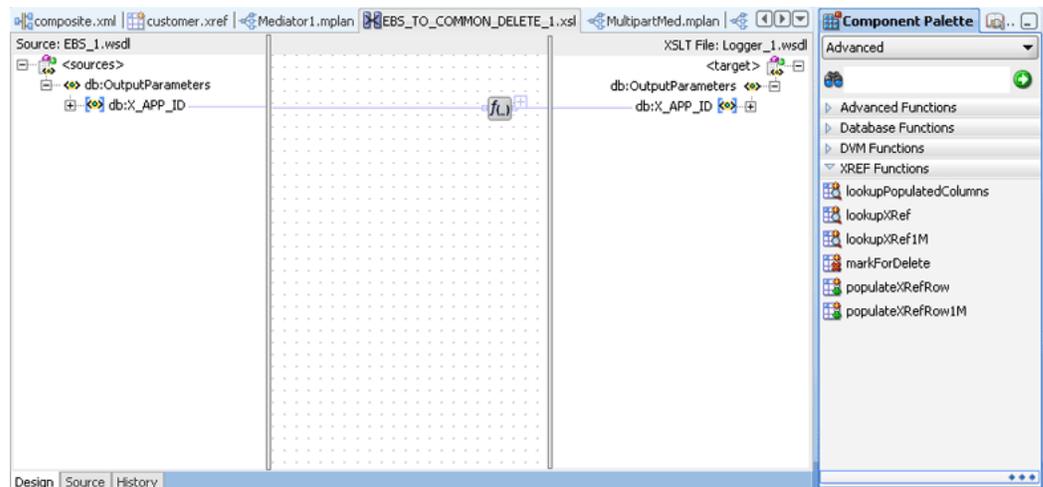
You can access the Expression builder dialog through Assign activity, XSL transformation, or Filter functionality of a BPEL service component or a Mediator service component. [Figure 47–2](#) shows how you can select the cross reference functions in the Expression builder dialog.

Figure 47–2 Expression Builder Dialog with Cross Reference Functions



The XSLT Mapper dialog is displayed when you create an XSL file to transform data from one XML schema to another. [Figure 47–3](#) shows how you can select the cross reference functions in the XSLT Mapper dialog.

Figure 47–3 XSLT Mapper Dialog with Cross Reference Functions



47.2 Creating and Modifying Cross Reference Tables

You can create cross references tables in a SOA composite application and then use it with a BPEL service component or a Mediator component during transformations.

47.2.1 Creating a Cross Reference Table

To create a cross reference table:

1. In Oracle JDeveloper, select the SOA project in which you want to create the cross reference.
2. Right-click the project and select **New**.
The New Gallery dialog is displayed.
3. Select **SOA Tier** from Categories and then select **Transformations**.
4. Select **Cross Reference(XREF)** from Items.
5. Click **OK**.

The Create Cross Reference(XREF) File dialog is displayed.

6. In the **File Name** field, specify the name of the cross reference file. For example, specify `Customer`.

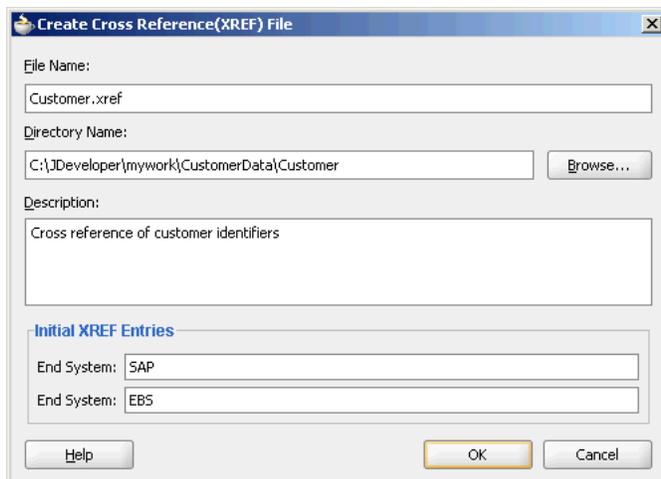
A cross reference name is used to identify a cross reference table uniquely. Two cross reference tables cannot have same name in the cross reference repository. The cross reference file name is the name of the cross reference table with an extension of `.xref`.

7. In the **Description** field, enter a description for the cross reference. For example, `Cross reference of Customer identifiers`.
8. In the **End System** fields, enter the end system names.

The end systems map to the cross reference columns in a cross reference table. For example, you can change the first end system name to `SAP` and second end system name to `EBS`. Each end system name must be unique within a cross reference.

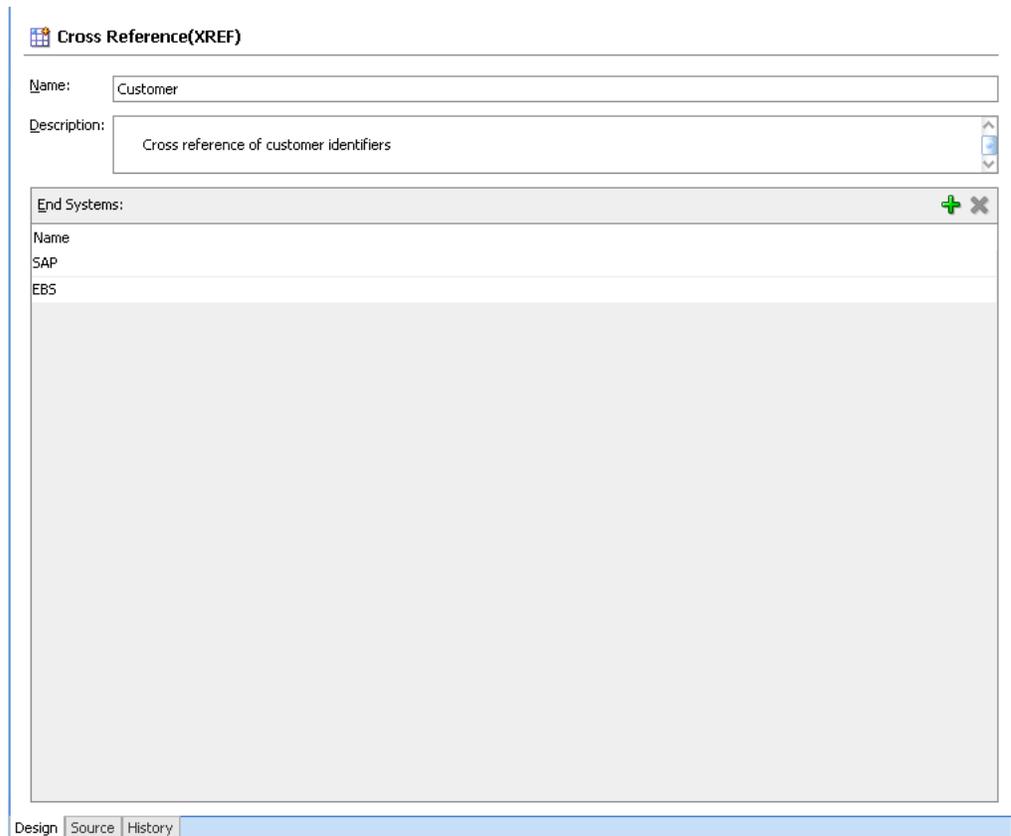
A sample Create Cross Reference(XREF) File dialog is displayed in [Figure 47-4](#).

Figure 47-4 Create Cross Reference(XREF) File Dialog



9. Click **OK**.

The Cross Reference(XREF) editor is displayed, as shown in [Figure 47-5](#). You can use this editor to modify the cross reference.

Figure 47–5 Cross Reference Editor

47.2.1.1 What Happens When You Create a Cross Reference

A file with extension `.xref` gets created and appears in the Application Navigator. All `.xref` files are based on the schema definition (XSD) file shown in [Example 47–1](#).

Example 47–1 Cross Reference XSD File

```
<?xml version="1.0" encoding="UTF-8" ?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://xmlns.oracle.com/xref"
  xmlns:tns="http://xmlns.oracle.com/xref" elementFormDefault="qualified">
  <element name="xref" type="tns:xrefType"/>
  <complexType name="xrefType">
    <sequence>
      <element name="table">
        <complexType>
          <sequence>
            <element name="description" type="string" minOccurs="0"
              maxOccurs="1"/>
            <element name="columns" type="tns:columnsType" minOccurs="0"
              maxOccurs="1"/>
            <element name="rows" type="tns:rowsType" maxOccurs="1"
              minOccurs="0"/>
          </sequence>
          <attribute name="name" type="string" use="required"/>
        </complexType>
      </element>
    </sequence>
  </complexType>
```

```
<complexType name="columnsType">
  <sequence>
    <element name="column" minOccurs="1" maxOccurs="unbounded">
      <complexType>
        <attribute name="name" type="string" use="required"/>
      </complexType>
    </element>
  </sequence>
</complexType>

<complexType name="rowsType">
  <sequence>
    <element name="row" minOccurs="1" maxOccurs="unbounded">
      <complexType>
        <sequence>
          <element name="cell" minOccurs="1" maxOccurs="unbounded">
            <complexType>
              <attribute name="colName" type="string" use="required"/>
            </complexType>
          </element>
        </sequence>
      </complexType>
    </element>
  </sequence>
</complexType>
</schema>
```

47.2.2 Adding an End System to a Cross Reference Table

To add an End System to a cross reference table:

1. Click **Add**.
A new row is added.
2. Double-click the newly added row.
3. Enter the End System name. For example, SBL.

47.3 Populating Cross Reference Tables

A cross reference table needs be populated at runtime before using it. By default, the data is stored in the XREF_DATA table under the SOA infrastructure database schema. You can use the `xref:populateXRefRow` function to populate a cross reference column with a single value and the `xref:populateXRefRow1M` function to populate a cross reference column with multiple values.

Note: You can also store the data in a different database schema by configuring a data source in the following way:

- The JNDI name of the data source should be `jdbc/xref`.
 - The `ORACLE_HOME/rcu/integration/soainfra/sql/xref/createschema_xref_oracle.sql` file should be loaded to create the XREF_DATA table in this data source.
-
-

47.3.1 About xref:populateXRefRow Function

The `xref:populateXRefRow` function populates a cross reference column with a single value. The `xref:populateXRefRow` function returns a string value which is the cross reference value being populated. For example, as shown in [Table 47-3](#), the Order table has the following columns: EBS, Common, and SBL with values E100, 100, and SBL_001 respectively.

Table 47-3 Cross Reference Table with Single Column Values

EBS	Common	SBL
E100	100	SBL_001

The syntax of the `xref:populateXRefRow` function is as follows:

```
xref:populateXRefRow(xrefLocation as string, xrefReferenceColumnName as string,
  xrefReferenceValue as string, xrefColumnName as string, xrefValue as string, mode
  as string) as string
```

Parameters

- `xrefLocation`: The cross reference table URI.
- `xrefReferenceColumnName`: The name of the reference column.
- `xrefReferenceValue`: The value corresponding to reference column name.
- `xrefColumnName`: The name of the column to be populated.
- `xrefValue`: The value to be populated in the column.
- `mode`: The mode in which the `xref:populateXRefRow` function populates the column. You can specify any of the following values: ADD, LINK, or UPDATE. [Table 47-4](#) describes these modes.

Table 47-4 xref:populateXRefRow Function Modes

Mode	Description	Exception Reasons
ADD	<p>Adds the reference value and the value to be added.</p> <p>For example, <code>xref:populateXRefRow("customers.xref", "EBS", "EBS100", "Common", "CM001", "ADD")</code> adds the reference value EBS100 in the ESB reference column and value CM001 in the Common column.</p>	<p>Exception can occur because of the following reasons:</p> <ul style="list-style-type: none"> ■ The specified cross reference table is not found. ■ The specified columns are not found. ■ The values provided are empty. ■ The value being added is not unique across that column for that table. ■ The column for that row already contains a value. ■ The reference value exists.

Table 47-4 (Cont.) xref:populateXRefRow Function Modes

Mode	Description	Exception Reasons
LINK	Adds the cross reference value corresponding to the existing reference value. For example, <code>xref:populateXRefRow("customers.xref", "Common", "CM001", "SBL", "SBL_001", "LINK")</code> links the value CM001 in the Common column to the SBL_001 value in the SBL column.	Exception can occur because of the following reasons: <ul style="list-style-type: none"> ■ The specified cross reference table is not found. ■ The specified columns are not found. ■ The values provided are empty. ■ The reference value is not found. ■ The value being linked exists in that column for that table.
UPDATE	Updates the cross reference value corresponding to an existing reference column-value pair. For example, <code>xref:populateXRefRow("customers.xref", "SBL", "SBL_001", "SBL", "SBL_1001", "Update")</code> updates the value SBL_001 in the SBL column to value SBL_1001.	Exception can occur because of the following reasons: <ul style="list-style-type: none"> ■ The specified cross reference table is not found. ■ The specified columns are not found. ■ The values provided are empty. ■ Multiple values are found for the column being updated. ■ The reference value is not found. ■ The column for that row does not have a value.

Note: The mode parameter values are case-sensitive and should be specified in the upper case only as shown in [Table 47-4](#).

[Table 47-5](#) describes the `xref:populateXRefRow` function modes and exception conditions for these modes.

Table 47-5 xref:populateXRefRow Function Results with Different Modes

Mode	Reference Value	Value to be Added	Result
ADD	Absent	Absent	Success
	Present	Absent	Exception
	Present	Present	Exception
LINK	Absent	Absent	Exception
	Present	Absent	Success
	Present	Present	Exception

Table 47-5 (Cont.) xref:populateXRefRow Function Results with Different Modes

Mode	Reference Value	Value to be Added	Result
UPDATE	Absent	Absent	Exception
	Present	Absent	Exception
	Present	Present	Success

47.3.2 About xref:populateXRefRow1M Function

Two values in an end system can correspond to a single value in another system. In such a scenario, you should use the `xref:populateXRefRow1M` function to populate a cross reference column with a value. For example, as shown in [Table 47-6](#), `SAP_001` and `SAP_0011` values refer to one value of the EBS and the SBL application. To populate the columns such as `SAP` column, you can use the `xref:populateXRefRow1M` function.

Table 47-6 Cross Reference Table with Multiple Column Values

SAP	EBS	SBL
SAP_001	EBS_1001	SBL001
SAP_0011		
SAP_002	EBS_1002	SBL002

The syntax of the `xref:populateXRefRow1M` function is as follows:

```
xref:populateXRefRow1M(xrefLocation as string, xrefReferenceColumnName as string,
  xrefReferenceValue as string, xrefColumnName as string, xrefValue as string, mode
  as string) as string
```

Parameters

- `xrefLocation`: The cross reference URI.
- `xrefReferenceColumnName`: The name of the reference column.
- `xrefReferenceValue`: The value corresponding to reference column name.
- `xrefColumnName`: The name of the column to be populated.
- `xrefValue`: The value to be populated in the column.
- `mode`: The mode in which the `xref:populateXRefRow` function populates the column. You can specify either of the following values: `ADD` or `LINK`. [Table 47-7](#) describes these modes:

Table 47–7 xref:populateXRefRow1M Function Modes

Mode	Description	Exception Reasons
ADD	Adds the reference value and the value to be added. For example, <code>xref:populateXRefRow1M("customers.xref", "EBS", "EBS_1002", "SAP", "SAP_0011", "ADD")</code> adds the reference value <code>EBS_1002</code> in the reference column <code>EBS</code> and the value <code>SAP_0011</code> in the <code>SAP</code> column.	Exception can occur because of the following reasons: <ul style="list-style-type: none"> ▪ The specified cross reference table is not found. ▪ The specified columns are not found. ▪ The values provided are empty. ▪ The value being added is not unique across that column for that table. ▪ The reference value exists.
LINK	Adds the cross reference value corresponding to the existing reference value. For example, <code>xref:populateXRefRow1M("customers.xref", "EBS", "EBS_1002", "SAP", "SAP_002", "LINK")</code> links the value <code>SAP_002</code> in the <code>SAP</code> column to the <code>EBS_1002</code> value in the <code>EBS</code> column.	Exception can occur because of the following reasons: <ul style="list-style-type: none"> ▪ The specified cross reference table is not found. ▪ The specified columns are not found. ▪ The values provided are empty. ▪ The reference value is not found. ▪ The value being added is not unique across the column for that table.

Table 47–8 describes the `xref:populateXRefRow1M` function modes and exception conditions for these modes.

Table 47–8 xref:populateXRefRow1M Function Results with Different Modes

Mode	Reference Value	Value to be Added	Result
ADD	Absent	Absent	Success
	Present	Absent	Exception
	Present	Present	Exception
LINK	Absent	Absent	Exception
	Present	Absent	Success
	Present	Present	Exception

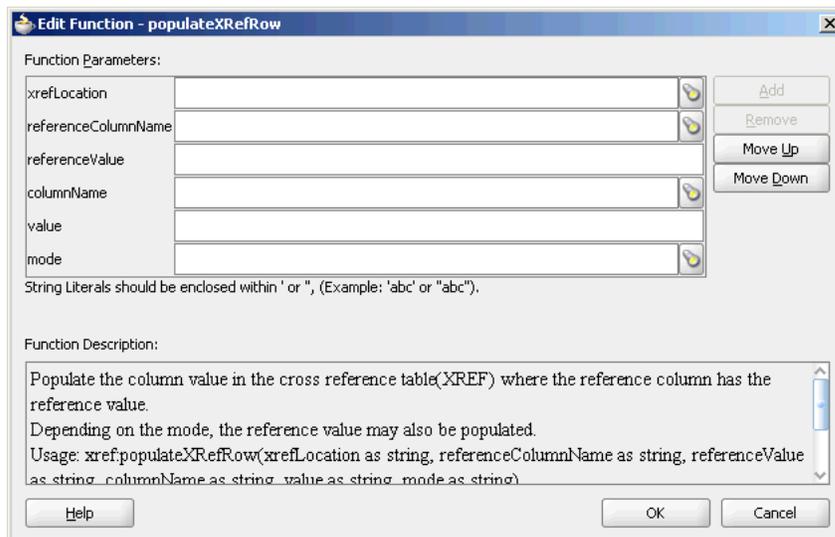
47.3.3 How to Populate a Column of a Cross Reference Table

To populate a column of a cross reference table:

1. In the XSLT Mapper window, expand the trees in the Source and Target panes.
2. Drag and drop the source element to the target element.

3. In the Component Palette, select **Advanced**.
4. Select **XREF Functions**.
5. Drag and drop the **populateXRefRow** function to the line that connects the source object to the target object.
A populateXRefRow icon appears on the connecting line.
6. Double-click the **populateXRefRow** icon.
The Edit Function – populateXRefRow dialog is displayed, as shown in [Figure 47–6](#).

Figure 47–6 Edit Function – populateXRefRow Dialog



7. Specify the following values for the fields in the Edit Function – populateXRefRow dialog:
 - a. In the **xrefLocation** field, enter the location URI of the cross reference file.
Click **Browse** to the right of the **xrefLocation** field to select the cross reference file. You can select an already deployed cross reference from MDS and also from a shared location in MDS using the Resource Palette.
 - b. In the **referenceColumnName** field, enter the name of the cross reference column.
Click **Browse** to the right of the **referenceColumnName** field to select a column name from the columns defined for the cross reference you previously selected.
 - c. In the **referenceValue** field, you can manually enter a value or press **Ctrl-Space** to launch XPath Building Assistant. Press the up and down keys to locate an object in the list and press enter to select that object.
 - d. In the **columnName** field, enter the name of cross reference column.
Click the **Browse** icon to the right of the **columnName** field to select a column name from the columns defined for the cross reference you previously selected.
 - e. In the **value** field, you can manually enter a value or press **Ctrl-Space** to launch XPath Building Assistant.

- f. In the **mode** field, enter a mode in which you want to populate the cross reference table column. For example, ADD.

You can also click **Browse** to select a mode. The Select Populate Mode dialog is displayed from which you can select a mode.

8. Click **OK**.

A populated Edit Function – populateXRefRow dialog is shown in [Figure 47–7](#).

Figure 47–7 Populated Edit Function – populateXRefRow Dialog

47.4 Looking Up Cross Reference Tables

After populating the cross reference table, you can use it to lookup for a value. The `xref:lookupXRef` and `xref:lookupXRef1M` functions enable you to look up a cross reference for single and multiple values respectively.

47.4.1 About `xref:lookupXRef` Function

You can use the `xref:lookupXRef` function to look up a cross reference column for a value that corresponds to a specific value in a reference column. For example, the following function looks up the `Common` column of the cross reference table, described in [Table 47–2](#), for a value corresponding to `SAP_001` value in `SAP` column.

```
xref:lookupXRef("customers.xref", "SAP", "SAP_001", "Common", true())
```

The syntax of the `xref:lookupXRef` function is as follows:

```
xref:lookupXRef(xrefLocation as string, xrefReferenceColumnName as string,
xrefReferenceValue as string, xrefColumnName as string, needAnException as
boolean) as string
```

Parameters

- `xrefLocation`: The cross reference URI.
- `xrefReferenceColumnName`: The name of the reference column.
- `xrefReferenceValue`: The value corresponding to reference column name.

- `xrefColumnName`: The name of the column to be looked up for the value.
- `needAnException`: When value is set to `true`, an exception is thrown if the value is not found, else an empty value is returned.

Exception Reasons

At runtime, an exception can occur because of the following reasons:

- The cross reference table with the given name is not found.
- The specified column names are not found.
- The specified reference value is empty.
- If multiple values are found.

47.4.2 About `xref:lookupXRef1M` Function

You can use the `xref:lookupXRef1M` function to look up a cross reference column for multiple values corresponding to a specific value in a reference column. The `xref:lookupXRef1M` function returns a node-set containing the multiple nodes. Each node in the node-set contains a value.

For example, the following function looks up the `SAP` column of [Table 47-6](#) for multiple values corresponding to `EBS_1001` value in the `EBS` column:

```
xref:lookupXRef1M("customers.xref", "EBS", "EBS_1001", "SAP", true())
```

The syntax of the `xref:lookupXRefRow1M` function is as follows:

```
xref:lookupXRef1M(xrefLocation as String, xrefReferenceColumnName as String,
  xrefReferenceValue as String, xrefColumnName as String, needAnException as
  boolean) as node-set
```

Parameters

- `xrefLocation`: The cross reference URI.
- `xrefReferenceColumnName`: The name of the reference column.
- `xrefReferenceValue`: The value corresponding to reference column name.
- `xrefColumnName`: The name of the column to be looked up for the value.
- `needAnException`: If this value is set to `true`, then an exception is thrown when the referenced value is not found. Else, an empty node-set is returned.

Example

Consider the `Order` table with the following three columns: `Siebel`, `Billing1`, `Billing2`

Table 47-9 *Order Table*

Siebel	Billing1	Billing2
100	101	102
110		111
		112

For 1:1 mapping, the

```
xref:lookupPopulatedColumns("Order", "Siebel", "100", "false")
method returns
```

```
<column name="BILLING1">101</column>
<column name="BILLING2">102</column>
```

In this case, both the columns, `Billing1` and `Billing2` are populated.

For 1:M mapping, the

```
xref:lookupPopulatedColumns("Order", "Siebel", "110", "false")
returns
```

```
<column name="BILLING2">111</column>
<column name="BILLING2">112</column>
```

In this case, `Billing1` is not populated.

Exception Reasons

An exception can occur because of the following reasons:

- The cross reference table with the given name is not found.
- The specified column names are not found.
- The specified reference value is empty.

47.4.3 About `xref:lookupPopulatedColumns` Function

You can use the `xref:lookupPopulatedColumns` function to look up all the populated columns for a given cross reference table, a cross reference column and a specific values. The `xref:lookupPopulatedColumns` function returns a nodeset with each node containing a column name and the corresponding value.

The syntax of the `xref:LookupPopulatedColumns` function is as follows:

```
xref:LookupPopulatedColumns(xrefTableName as String,xrefColumnName as
String,xrefValue as String,needAnException as boolean)as node-set
```

Parameters

- `xrefTableName`: The name of the reference table.
- `xrefColumnName`: The name of the reference column.
- `xrefValue`: The value corresponding to reference column name.
- `needAnException`: If this value is set to `true`, then an exception is thrown when no value is found in the referenced column. Otherwise, an empty node-set is returned.

Exception Reasons

An exception can occur because of the following reasons:

- The cross reference table with the given name is not found.
- The specified column names are not found.
- The specified reference value is empty.

47.4.4 How to Look Up a Cross Reference Table for a Value

To look up a cross reference table column:

1. In the XSLT Mapper dialog, expand the trees in the Source and Target panes.

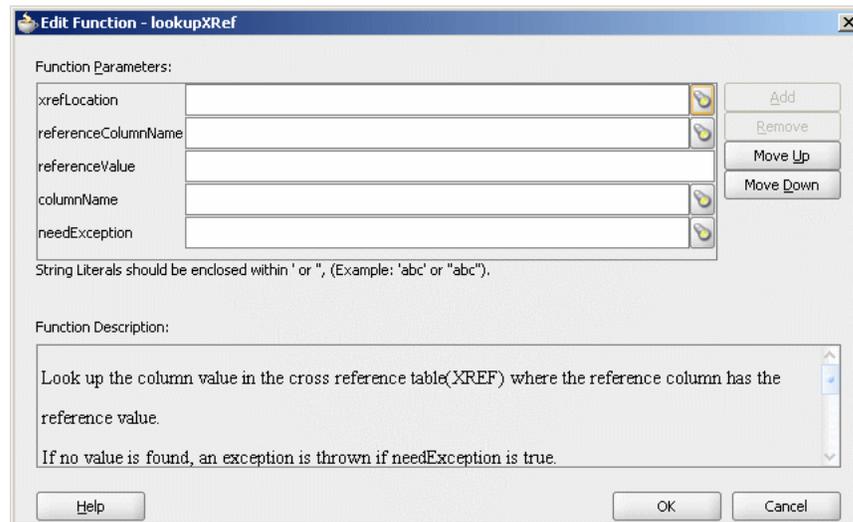
2. Drag and drop the source element to the target element.
3. In the Component Palette, select **Advanced**.
4. Select **XREF Functions**.
5. Drag and drop the **lookupXRef** function to the line that connects the source object to the target object.

A lookupXRef icon appears on the connecting line.

6. Double-click the **lookupXRef** icon.

The Edit Function – lookupXRef dialog is displayed, as shown in [Figure 47–8](#).

Figure 47–8 Edit Function – lookupXRef Dialog

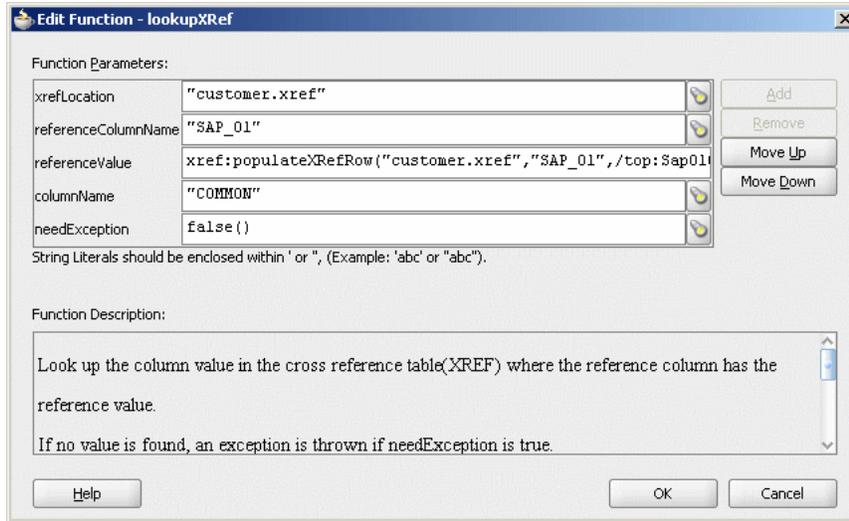


7. Specify the following values for the fields in the Edit Function – lookupXRef dialog:
 - a. In the **xrefLocation** field, enter the location URI of the cross reference file.
Click **Browse** to the right of the **xrefLocation** field to select the cross reference file. You can select an already deployed cross reference from MDS and also from shared location in MDS by using the Resource Palette.
 - b. In the **referenceColumnName** field, enter the name of the cross reference column.
Click **Browse** to the right of the **referenceColumnName** field to select a column name from the columns defined for the cross reference you previously selected.
 - c. In the **referenceValue** field, you can manually enter a value or press **Ctrl-Space** to use the XPath Building Assistant. Press the up and down keys to locate an object in the list and press enter to select that object.
 - d. In the **columnName** field, enter the name of the cross reference column.
Click **Browse** to the right of the **columnName** field to select a column name from the columns defined for the cross reference you previously selected.
 - e. Click **Browse** to the right of **needException** field. The Need Exception dialog is displayed. Select **YES** to raise an exception if no value is found else select **No**.

8. Click OK.

A populated Edit Function – lookupXRef dialog is shown in Figure 47–9.

Figure 47–9 Populated Edit Function – lookupXRef Dialog



47.5 Deleting a Cross Reference Table Value

You can use the `xref:markForDelete` function to delete a value in a cross reference table. The value in the column is marked as deleted. This function returns `true` if deletion is successful else returns `false`.

Any column value marked for delete is treated as if the value does not exist. Therefore, you can populate the same column with `xref:populateXRefRow` function in ADD mode.

Note: Using a column value marked for delete as a reference value in the LINK mode of `xref:populateXRefRow` function, raises an error.

A cross reference table row should have at least two mappings. if you have only two mappings in a row and you mark one value for delete, then the value in another column is also deleted.

The syntax for the `xref:markForDelete` function is as follows:

```
xref:markForDelete(xrefTableName as string, xrefColumnName as string,
xrefValueToDelete as string) return as boolean
```

Parameters

- `xrefTableName`: The cross reference table name.
- `xrefColumnName`: The name of the column from which you want to delete a value.
- `xrefValueToDelete`: The value to be deleted.

Exception Reasons

An exception can occur because of the following reasons:

- The cross reference table with the given name is not found.
- The specified column name is not found.
- The specified value is empty.
- The specified value is not found in the column.
- Multiple values are found.

47.5.1 How to Delete a Cross Reference Table Value

To delete a cross reference table value:

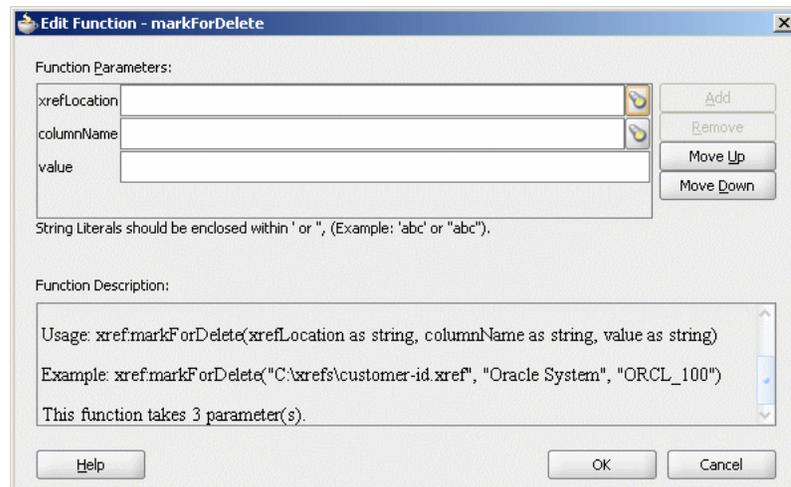
1. In the XSLT Mapper window, expand the trees in the Source and Target panes.
2. Drag and drop the source element to the target element.
3. In the Component Palette, select **Advanced**.
4. Select **XREF Functions**.
5. Drag and drop the **markForDelete** function to the line that connects the source object to the target object.

A markForDelete icon appears on the connecting line.

6. Double-click the **markForDelete** icon.

The Edit Function – markForDelete dialog is displayed, as shown in [Figure 47–10](#).

Figure 47–10 Edit Function – markForDelete Dialog



7. Specify the following values for the fields in the Edit Function – markForDelete dialog:

- a. In the **xrefLocation** field, enter the location URI of the cross reference file.

Click the flashlight icon to the right of the **xrefLocation** field to select the cross reference file. You can select an already deployed cross reference from MDS and also from shared location in MDS by using the Resource Palette.

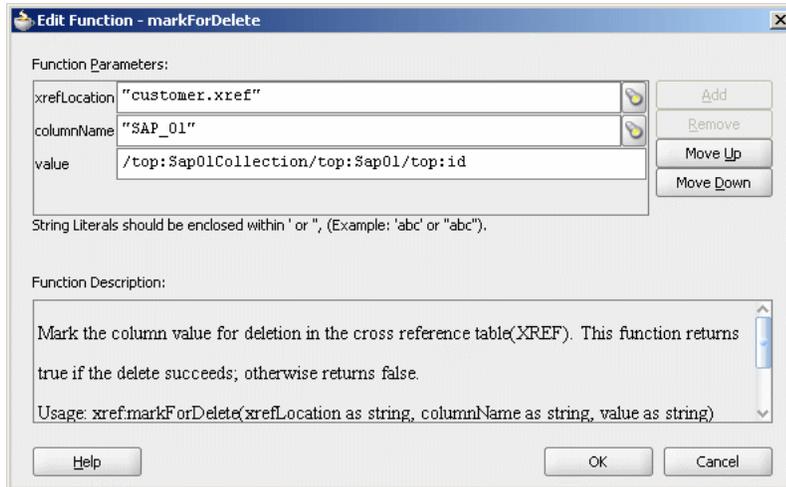
- b. In the **columnName** field, enter the name of cross reference table column.

Click the flashlight icon to the right of the **columnName** field to select a column name from the columns defined for the cross reference you previously selected.

- c. In the **Value** field, you can manually enter a value or press **Ctrl-Space** to launch XPath Building Assistant. Press the up and down keys to locate an object in the list and press enter to select that object.

A populated Edit Function – markForDelete dialog is shown in [Figure 47–11](#).

Figure 47–11 Populated Edit Function – markForDelete Dialog

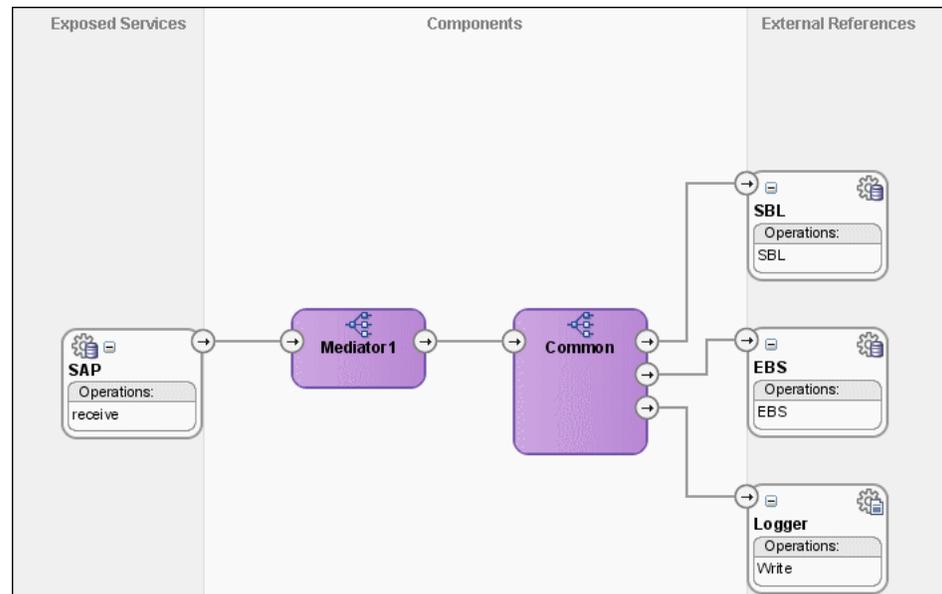


- 8. Click **OK**.

47.6 Creating and Running Cross Reference Use Case

The cross reference use case implements an integration scenario between Oracle EBS, SAP and Siebel instances. In this use case, when an insert, update, or delete operation is performed on the SAP_01 table, the corresponding data is inserted or updated in the EBS and SBL tables. [Figure 47–12](#) provides an overview of this use case.

Figure 47-12 XrefCustApp Use Case in SOA Composite Editor



For downloading the sample files mentioned in this section, visit the following URL:

http://www.oracle.com/technology/sample_code/products/mediator

47.6.1 Step-By-Step Instructions for Creating the Use Case

This section provides the design-time tasks for creating, building, and deploying your SOA Composite application. These tasks should be performed in the order in which they are presented.

- Section 47.6.1.1, "Task 1: Configuring Oracle Database and Database Adapter"
- Section 47.6.1.2, "Task 2: Creating an Oracle JDeveloper Application and Project"
- Section 47.6.1.3, "Task 3: Creating a Cross Reference"
- Section 47.6.1.4, "Task 4: Creating a Database Adapter Service"
- Section 47.6.1.5, "Task 5: Creating EBS and SBL External References"
- Section 47.6.1.6, "Task 6: Creating Logger External Reference"
- Section 47.6.1.7, "Task 7: Creating Mediator Components"
- Section 47.6.1.8, "Task 8: Specifying Routing Rules for Mediator Component"
- Section 47.6.1.9, "Task 9: Specifying Routing Rules for Common Mediator"
- Section 47.6.1.10, "Task 10: Configuring Oracle Application Server Connection"
- Section 47.6.1.11, "Task 11: Deploying the Composite Application"

47.6.1.1 Task 1: Configuring Oracle Database and Database Adapter

To configure Oracle Database and the Database adapter

1. You need SCOTT database account with password TIGER for this use case. You must ensure that the SCOTT account is unlocked.

You can log in as SYSDBA and then run the `setup_user.sql` script available in the `XrefOrderApp1M/sql` folder to unlock the account.

2. Run the `create_schema.sql` script available in the `XrefOrderApp1M/sql` folder to create the tables required for this use case.
3. Run the `create_app_procedure.sql` script available in the `XrefOrderApp1M/sql` folder to create a procedure that simulates the various applications participating in this integration.
4. Run the `createschema_xref_oracle.sql` script available in the `OH/rcu/integration/soainfra/sql/xref/` folder to create a Cross Reference table to store runtime Cross Reference data.
5. Copy the `ra.xml` and `weblogic-ra.xml` files from `$BEAHOME/META-INF` to the newly created directory called `META-INF` on your computer.
6. Edit the `weblogic-ra.xml` file, available in the `$BEAHOME/META-INF` folder as follows:

- Modify the property to `xDataSourceName` as follows:

```
<property>
  <name>xDataSourceName</name>
  <value>jdbc/DBConnection1</value>
</property>
```

- Modify the `jndi-name` as follows:

```
<jndi-name> eis/DB/DBConnection1</jndi-name>
```

This sample uses `eis/DB/DBConnection1` to poll SAP table for new messages and to connect to the procedure that simulates Oracle EBS and Siebel instances.

7. Package the `ra.xml` and `weblogic-ra.xml` files as a RAR file and deploy the RAR file by using the Weblogic console.
8. Create a data source using the Weblogic console with the following values:
 - **jndi-name**=`jdbc/DBConnection1`
 - **user**=`scott`
 - **password**=`tiger`
 - **url**=`jdbc:oracle:thin:@host:port:service`
 - **connection-factory**
factory-class=`oracle.jdbc.pool.OracleDataSource`
9. Create a data source using the Weblogic console with the following values:
 - **jndi-name**=`jdbc/xref`
 - **user**=`scott`
 - **password**=`tiger`
 - **url**=`jdbc:oracle:thin:@host:port:service`
 - **connection-factory**
factory-class=`oracle.jdbc.pool.OracleDataSource`

47.6.1.2 Task 2: Creating an Oracle JDeveloper Application and Project

To create an application and a project:

1. In Oracle JDeveloper, click **File** and select **New**.
The New Gallery dialog appears.

2. In the New Gallery, expand the **General** node, and select the **Applications** category.
3. In the **Items** list, select **SOA Application** and click **OK**.
The Create SOA Application wizard appears.
4. In the **Application Name** field, enter `XrefCustApp`, and then click **Next**.
The **Name your SOA project** screen appears.
5. In the **Project Name** field, enter `XrefCustApp` and click **Next**.
The **Configure SOA settings for the SOA project** screen appears.
6. In the Composite Template list, select **Empty Composite** and then click **Finish**.
The Applications Navigator of Oracle JDeveloper is updated with the new application and project and the Design tab contains, a blank palette.
7. From the **File** menu, click **Save All**.

47.6.1.3 Task 3: Creating a Cross Reference

After creating an application and a project for the use case, you must create a cross reference table.

To create a cross reference table:

1. In the Application Navigator, right-click the **XrefCustApp** project and select **New**.
2. In the New Gallery dialog, expand the **SOA Tier** node, and then select the **Transformations** category.
3. In the Items list, select **Cross Reference(XREF)** and click **OK**.
The Create Cross Reference(XREF) File dialog is displayed.
4. In the **File Name** field, enter `customer.xref`.
5. In the **End System** fields, enter `SAP_01` and `EBS_i76`.
6. Click **OK**.
The Cross Reference Editor is displayed.
7. Click **Add**.
A new row is added.
8. Enter `SBL_78` as the End System name in the newly added row.
9. Click **Add** and enter `Common` as End System name.
The Cross Reference Editor would appear as shown in [Figure 47-13](#).

Figure 47–13 Customer Cross Reference



10. From the **File** menu, click **Save All** and close the Cross Reference Editor.

47.6.1.4 Task 4: Creating a Database Adapter Service

To create a Database adapter service:

1. In the Component Palette, select **SOA**.
2. Select **Database Adapter** and drag it to the Exposed Services design area.
The Adapter Configuration wizard Welcome page is displayed.
3. Click **Next**.
The Service Name page is displayed.
4. In the **Service Name** field, enter **SAP**.
5. Click **Next**.
The Service Connection page is displayed.
6. In the **Application Connection** field, select **DBConnection1**.
7. In the **JNDI Name** field, enter **eis/DB/DBConnection1**.
8. Click **Next**.
The Operation Type page is displayed.
9. Select **Poll for New or Changed Records in a Table** and click **Next**.
The Select Table page is displayed.
10. Click **Import Tables**.
The Import Tables dialog is displayed.
11. Select **Scott** from Schema.
12. In the **Name Filter** field, enter **%SAP%** and click **Query**.
The **Available** field is populated with **SAP_01** table name.
13. Double-click **SAP_01**.

The **selected** field is populated with SAP_01.

14. Click **OK**.

The Select Table page now contains the SAP_01 table.

15. Select **SAP_01** and click **Next**.

The Define Primary Key page is displayed.

16. Select **ID** as primary key and click **Next**.

The Relationships page is displayed.

17. Click **Next**.

The Attribute Filtering page is displayed.

18. Click **Next**.

The After Read page is displayed.

19. Select **Update a Field in the [SAP_01] Table (Logical Delete)** and click **Next**.

The Logical Delete page is displayed.

20. In the **Logical Delete** field, select **LOGICAL_DEL**.

21. In the **Read Value** field, enter **Y**.

22. In the **Unread Value** field, enter **N**.

[Figure 47–14](#) shows the Logical Delete page of the Adapter Configuration Wizard.

Figure 47–14 Logical Delete Page: Adapter Configuration Wizard

Adapter Configuration Wizard - Step 10 of 13

Logical Delete

Specify the field that should be updated to logically delete the row, and the value to insert in the field to indicate that the row has been read. You can also optionally specify values that indicate if a row is Unread or Reserved.

Logical Delete Field: LOGICAL_DEL

Read Value: Y

Unread Value: N

Reserved Value:

Help < Back Next > Finish Cancel

23. Click **Next**.

The Polling Options page is displayed.

24. Click **Next**.

The Define Selection Criteria page is displayed.

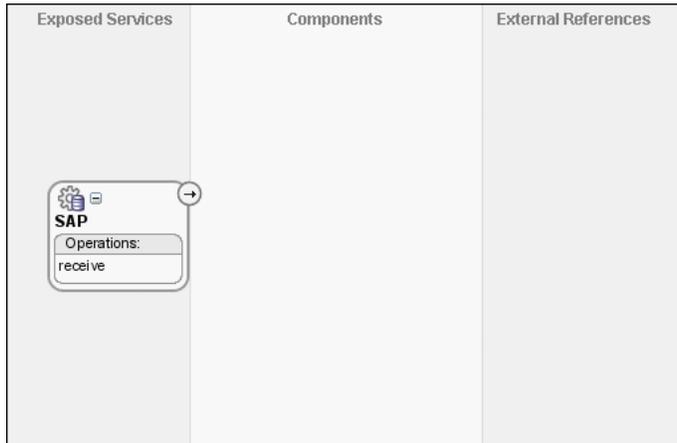
25. Click **Next**.

The Finish page is displayed.

26. Click **Finish**.

A Database adapter service *SAP* is created, as shown in [Figure 47–15](#).

Figure 47–15 SAP Database Adapter Service in SOA Composite Editor



27. From the **File** menu, click **Save All**.

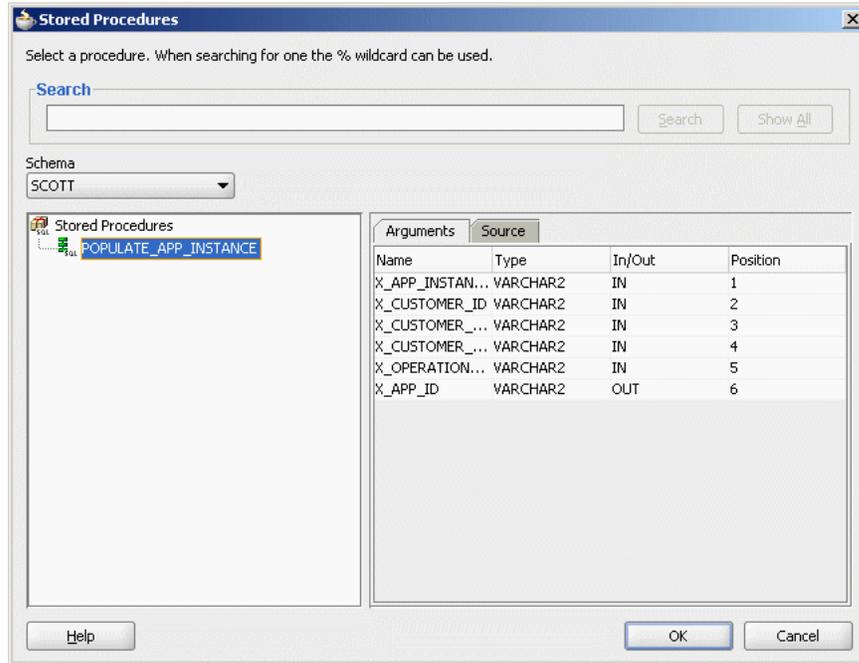
47.6.1.5 Task 5: Creating EBS and SBL External References

To create external references named EBS and SBL:

1. In the Component Palette, select **SOA**.
2. Select **Database Adapter** and drag it to the External References design area.
The Adapter Configuration wizard Welcome page is displayed.
3. Click **Next**.
The Service Name page is displayed.
4. In the **Service Name** field, enter **EBS**.
5. Click **Next**.
The Service Connection page is displayed.
6. In the **Application Connection** field, select **DBConnection1**.
7. In the **JNDI Name** field, enter `eis/DB/DBConnection1`.
8. Click **Next**.
The Operation Type page is displayed.
9. Select **Call a Stored Procedure or Function** and click **Next**.
The Specify Stored Procedure page is displayed.
10. Select **Scott** from Schema.
11. Click **Browse**.
The Stored Procedures dialog is displayed.

12. Select POPULATE_APP_INSTANCE as shown in Figure 47-16.

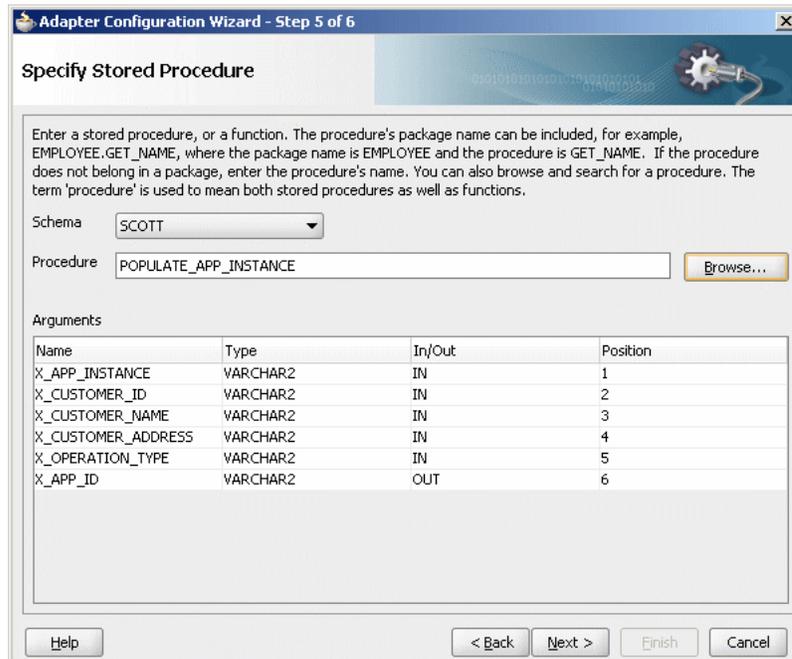
Figure 47-16 Stored Procedure Dialog



13. Click OK.

The Specify Stored Procedure page appears as shown in Figure 47-17.

Figure 47-17 Specify Stored Procedure Page of Adapter Configuration Wizard



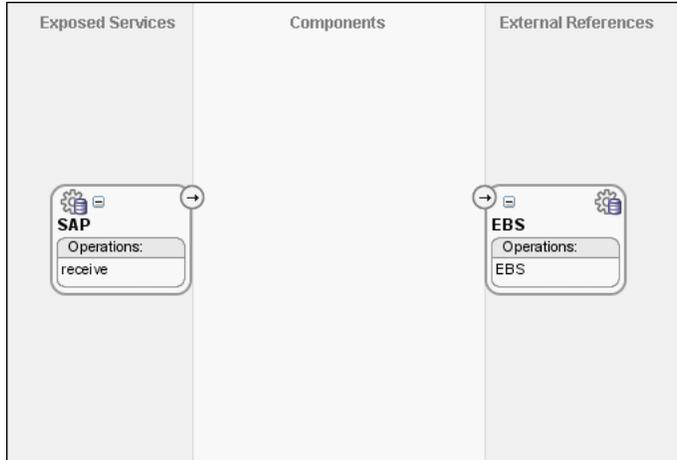
14. Click Next.

The Finish page is displayed.

15. Click **Finish**.

Figure 47-18 shows the EBS reference in SOA Composite Editor.

Figure 47-18 EBS Reference in SOA Composite Editor

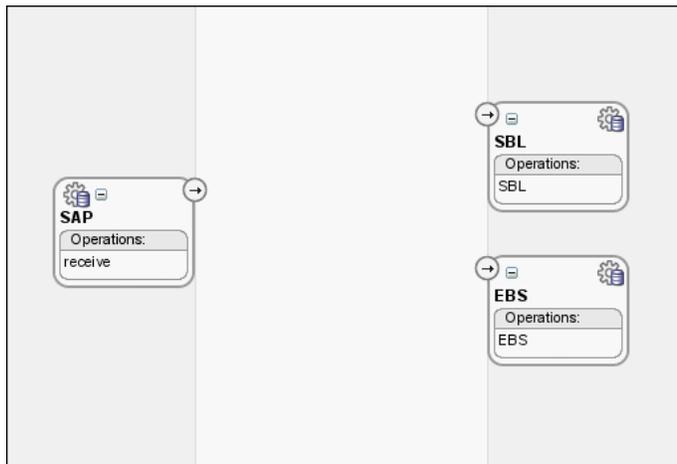


16. From the **File** menu, click **Save All**.

17. Repeat Step 2 through Step 16 to create another external references names SBL.

After completing this task, the SOA Composite Editor would appear as shown in Figure 47-19.

Figure 47-19 SBL Reference in SOA Composite Editor



47.6.1.6 Task 6: Creating Logger External Reference

To create a file adapter reference:

1. From the Component Palette, select **SOA**.
2. Select **File Adapter** and drag it to the External References design area.
The Adapter Configuration wizard Welcome page is displayed.
3. Click **Next**.

The Service Name page is displayed.

4. In the **Service Name** field, enter `Logger`.
5. Click **Next**.

The Operation page is displayed.

6. In the **Operation Type** field, select **Write File**.
7. Click **Next**.

The File Configuration page is displayed.

8. In the **Directory for Outgoing Files (physical path)** field, enter the name of the directory where you want to write the files.
9. In the **File Naming Convention** field, enter `output.xml` and click **Next**.

The Messages page is displayed.

10. Click **Search**.

The Type Chooser dialog is displayed.

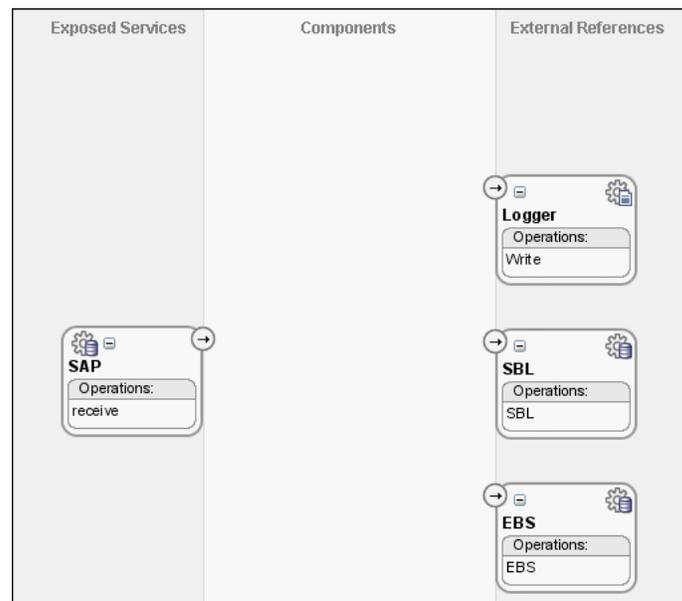
11. Navigate to **Type Explorer, Project Schema Files, SCOTT_POPULATE_APP_INSTANCE.xsd** and then select **OutputParameters**.
12. Click **OK**.
13. Click **Next**.

The Finish page is displayed.

14. Click **Finish**.

Figure 47–20 shows the `Logger` reference in the SOA Composite Editor.

Figure 47–20 *Logger Reference in SOA Composite Editor*



15. From the **File** menu, click **Save All**.

47.6.1.7 Task 7: Creating Mediator Components

To create a Mediator component:

1. Drag and drop a Mediator from Components Palette to the Components design area.

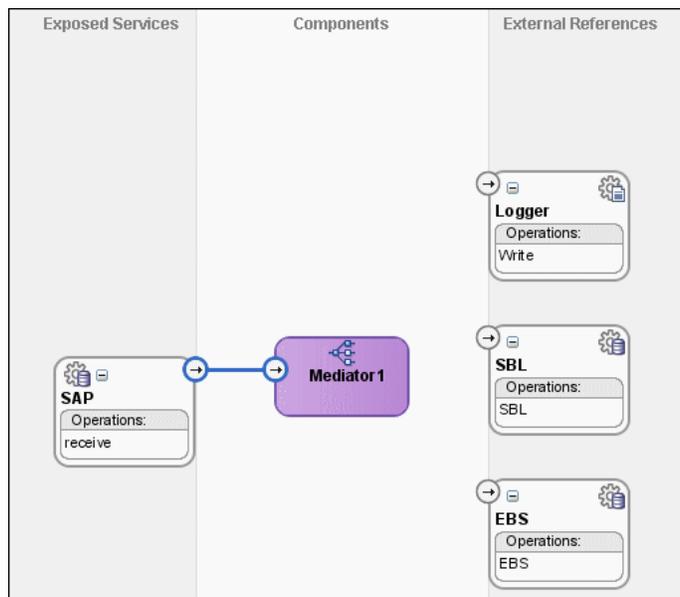
The Create Mediator dialog is displayed.

2. Select **Define Interface Later** from Template.
3. Click **OK**.

A Mediator with name `Mediator1` is created.

4. Connect the **SAP** service to the **Mediator1** as shown in [Figure 47–21](#).

Figure 47–21 SAP Service Connected to Mediator1



5. Click **Save All**.
6. Drag and drop another Mediator from Components Palette to the Components design area.
The Create Mediator dialog is displayed.
7. Select **Interface Definition From WSDL** from Template.
8. Deselect **Create Composite Service with SOAP Bindings**.
9. Click **Find Existing WSDLs** to the right of the **WSDL File** field.
10. Navigate to and then select the `Common.wsdl` file. The `Common.wsdl` file is available in the `Samples` folder.
11. Click **OK**.
12. Click **OK**.

A Mediator with name `Common` is created.

47.6.1.8 Task 8: Specifying Routing Rules for Mediator Component

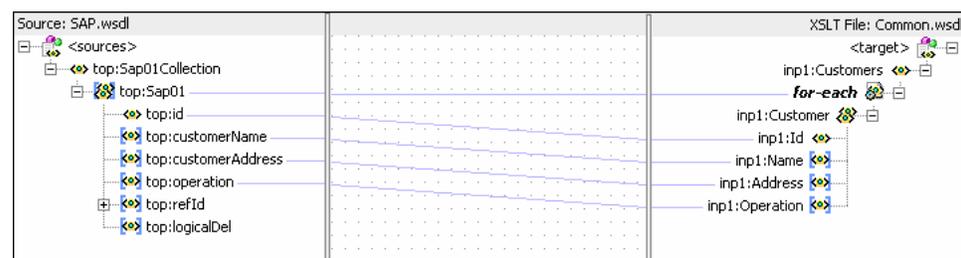
You must specify routing rules for following operations:

- Insert
- Update
- UpdateID
- Delete

To create routing rules for insert operation:

1. Double-click **Mediator1** Mediator.
The Mediator Editor is displayed.
2. In Routing Rules panel, click the **Create a new Routing Rule** icon.
The Target Type dialog is displayed.
3. Select **Service**.
The Target Services dialog is displayed.
4. Navigate to **XrefCustApp, Mediators, Common, Services, Common**.
5. Select **Insert** and click **OK**.
6. Click the **Filter** icon.
The Expression Builder dialog is displayed.
7. Enter the following expression in the **Expression** field:
`$in.Sap01Collection/top:Sap01Collection/top:Sap01/top:operation='INSERT'`
8. Click **OK**.
9. Click the **Transformation** icon next to the **Transform Using** field.
The Request Transformation map dialog is displayed.
10. Select **Create New Mapper File** and enter `SAP_TO_COMMON_INSERT.xml`.
11. Click **OK**.
A `SAP_TO_COMMON_INSERT.xml` tab is displayed.
12. Drag and drop the **top:SAP01** source element to the **inp1:Customer** target element.
The Auto Map Preferences dialog is displayed.
13. From the **During Auto Map** options, deselect **Match Elements Considering their Ancestor Names**.
14. Click **OK**.
The transformation is created as shown in [Figure 47-22](#).

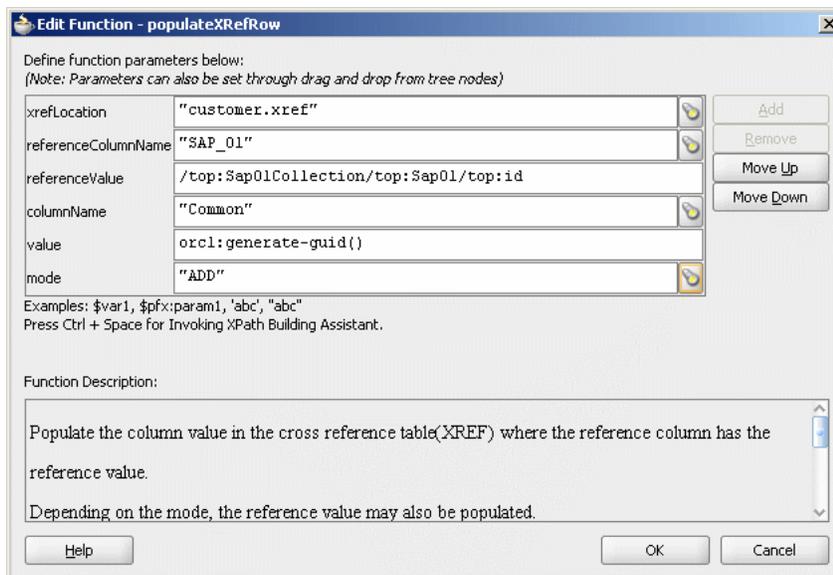
Figure 47-22 `SAP_TO_COMMON_INSERT.xml` Transformation



15. From the Components Palette, select **Advanced**.
16. Select **XREF Functions**.
17. Drag and drop the **populateXRefRow** function from Components Palette to the line connecting **top:id** and **inp1:id** elements.
18. Double-click the **populateXRefRow** icon.
The Edit Function-populateXRefRow dialog is displayed.
19. Click **Search** to the right of **xrefLocation** field.
The SCA Resource Lookup dialog is displayed.
20. Select **customer.xref** and click **OK**.
21. In the **referenceColumnName** field, enter "SAP_01" or click **Search** to select the column name.
22. In the **referenceValue** column, enter
/top:Sap01Collection/top:Sap01/top:id.
23. In the **columnName** field, enter "Common" or click **Search** to select the column name.
24. In the **value** field, enter `oraext:generate-guid()`.
25. In the **mode** field, enter "Add" or click **Search** to select this mode.

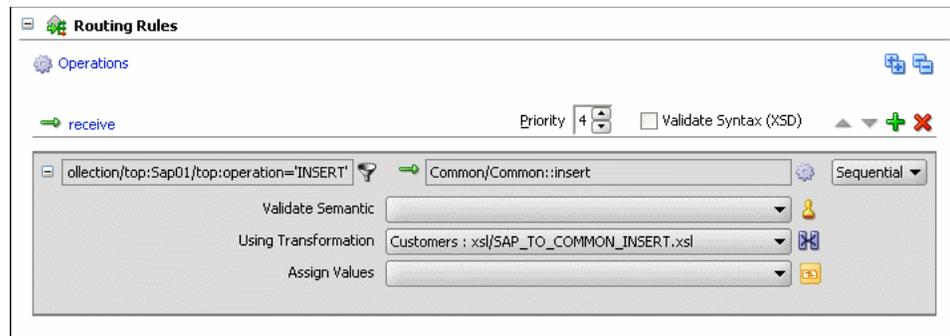
Figure 47–23 shows populated Edit Function – populateXRefRow dialog.

Figure 47–23 Edit Function – populateXRefRow Dialog: XrefCustApp Use Case



26. Click **OK**.
27. From the **File** menu, click **Save All** and close the SAP_TO_COMMON_INSERT.xml tab.

The Routing Rules panel would appear as shown in Figure 47–24.

Figure 47–24 Routing Rules Panel with Insert Operation**To create routing rules for update operation:**

Perform the following tasks to create routing rules for Update operation:

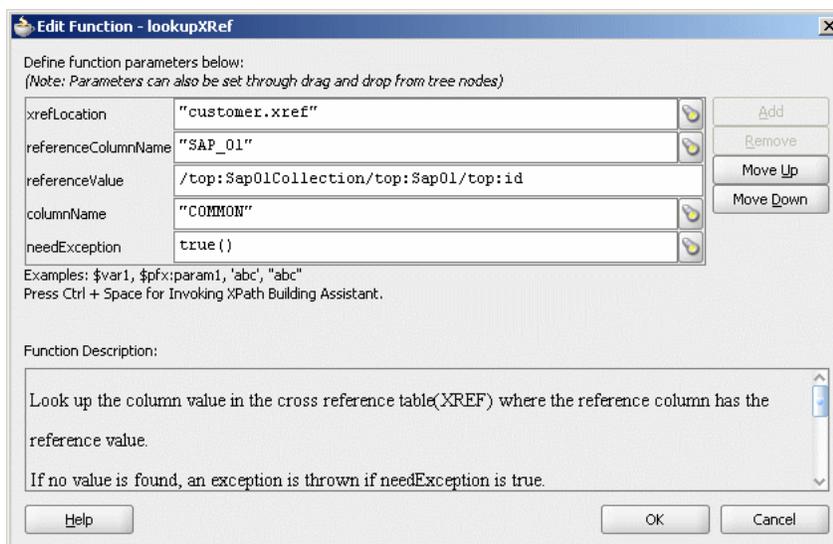
1. In Routing Rules panel, click the **Create a new Routing Rule** icon.
The Target Type dialog is displayed.
2. Select **Service**.
The Target Services dialog is displayed.
3. Navigate to **XrefCustApp, Mediators, Common, Services, Common**.
4. Select **Update** and click **OK**.
5. Click the **Filter** icon.
The Expression Builder dialog is displayed.
6. Enter the following expression in the **Expression** field:
`$in.Sap01Collection/top:Sap01Collection/top:Sap01/top:operation='UPDATE'`
7. Click **OK**.
8. Click the **Transformation** icon next to the **Transform Using** field.
The Request Transformation map dialog is displayed.
9. Select **Create New Mapper File** and enter `SAP_TO_COMMON_UPDATE.xsl`.
10. Click **OK**.
A `SAP_TO_COMMON_UPDATE.xsl` tab is displayed.
11. Drag and drop the **top:Sap01** source element to the **inp1:Customer** target element.
The Auto Map Preferences dialog is displayed.
12. Click **OK**.
13. From the **Components Palette**, select **Advanced**.
14. Select **XREF Functions**.
15. Drag and drop the **lookupXRef** function from Components Palette to the line connecting **top:id** and **inp1:id** elements.
16. Double-click the **lookupXRef** icon.
The Edit Function-lookupXRef dialog is displayed.
17. Click **Search** to the right of **xrefLocation** field.

The SCA Resource Lookup dialog is displayed.

18. Select **customer.xref** and click **OK**.
19. In the **referenceColumnName** field, enter "SAP_01" or click **Search** to select the column name.
20. In the **referenceValue** column, enter
/top:Sap01Collection/top:Sap01/top:id.
21. In the **columnName** field, enter "COMMON" or click **Search** to select the column name.
22. In the **needException** field, enter true () or click **Search** to select this mode.

Figure 47–25 shows populated Edit Function – loopXRef dialog.

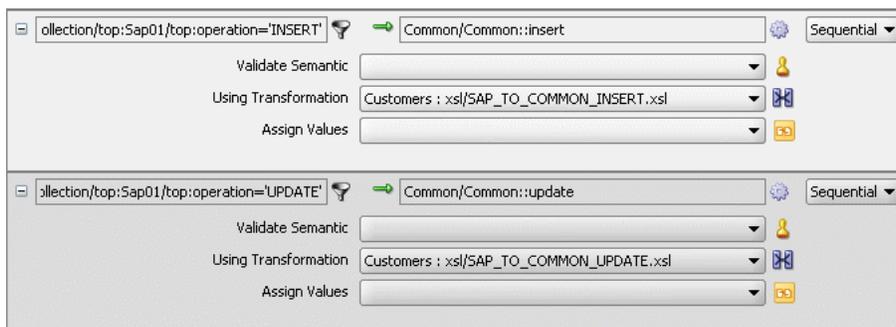
Figure 47–25 Edit Function – lookupXRef Dialog: XrefCustApp Use Case



23. Click **OK**.
24. From the **File** menu, click **Save All** and close the SAP_TO_COMMON_UPDATE.xsl tab.

The Routing Rules panel would appear as shown in Figure 47–26.

Figure 47–26 Insert Operation and Update Operation



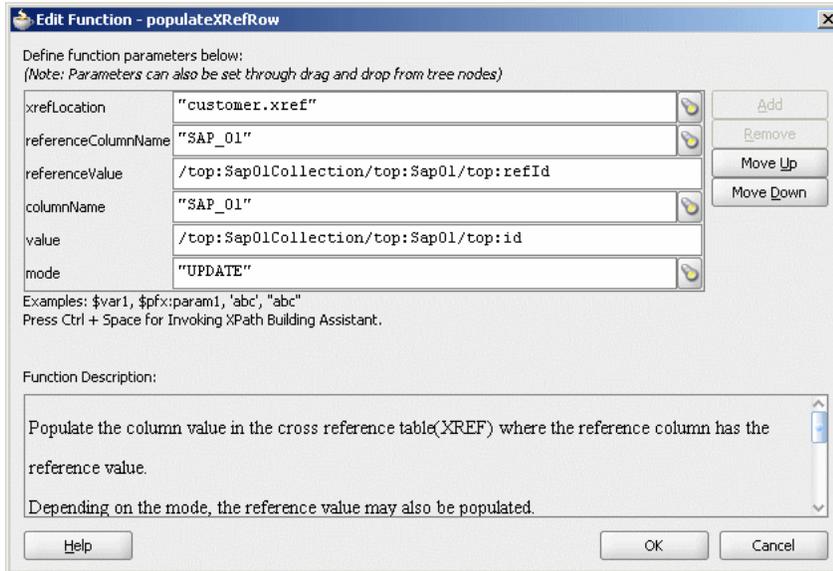
To create routing rules for updateID operation:

Perform the following tasks to create routing rules for UpdateID operation:

1. In Routing Rules panel, click the **Create a new Routing Rule** icon.
The Target Type dialog is displayed.
2. Select **Service**.
The Target Services dialog is displayed.
3. Navigate to **XrefCustApp, Mediators, Common, Services, Common**.
4. Select **updateid** and click **OK**.
5. Click the **Filter** icon.
The Expression Builder dialog is displayed.
6. Enter the following expression in the **Expression** field:
`$in.Sap01Collection/top:Sap01Collection/top:Sap01/top:operation = 'UPDATEID'`
7. Click **OK**.
8. Click the **Transformation** icon next to the **Transform Using** field.
The Request Transformation map dialog is displayed.
9. Select **Create New Mapper File** and enter **SAP_TO_COMMON_UPDATEID.xsl**.
10. Click **OK**.
A **SAP_TO_COMMON_UPDATEID.xsl** tab is displayed.
11. Drag and drop the **top:Sap01** source element to the **inp1:Customer** target element.
The Auto Map Preferences dialog is displayed.
12. Click **OK**.
13. From the **Components** Palette, select **Advanced**.
14. Select **XREF Functions**.
15. Drag and drop the **populateXRefRow** function from Components Palette to the line connecting **top:id** and **inp1:id** elements.
16. Double-click the **populateXRefRow** icon.
The Edit Function-populateXRefRow dialog is displayed.
17. Click **Search** to the right of **xrefLocation** field.
The SCA Resource Lookup dialog is displayed.
18. Select **customer.xref** and click **OK**.
19. In the **referenceColumnName** field, enter "SAP_01" or click **Search** to select the column name.
20. In the **referenceValue** column, enter
`/top:Sap01Collection/top:Sap01/top:refId`.
21. In the **columnName** field, enter "SAP_01" or click **Search** to select the column name.
22. In the **value** field, enter `/top:Sap01Collection/top:Sap01/top:Id`.
23. In the **mode** field, enter "UPDATE" or click **Search** to select this mode.

Figure 47–27 shows a populated Edit Function – populateXRefRow dialog.

Figure 47–27 Edit Function – populateXRefRow Dialog: XrefCustApp Use Case

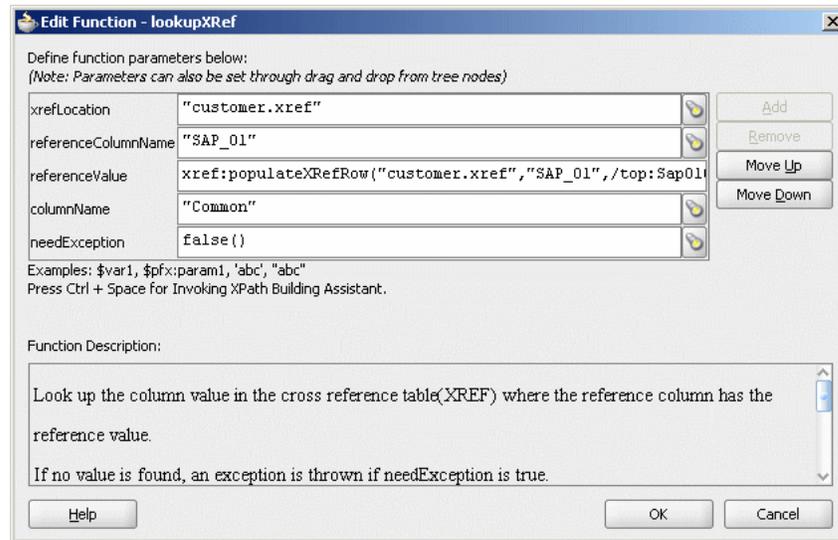


24. Drag and drop the **lookupXRef** function from Components Palette to the line connecting **top:id** and **inp1:id** elements.
25. Double-click the **lookupXRef** icon.
The Edit Function-lookupXRef dialog is displayed.
26. Click **Search** to the right of **xrefLocation** field.
The SCA Resource Lookup dialog is displayed.
27. Select **customer.xref** and click **OK**.
28. In the **referenceColumnName** field, enter "SAP_01" or click **Search** to select the column name.
29. In the **referenceValue** column, enter the following:

```
xref:populateXRefRow("customer.xref", "SAP_01", /top:Sap01Collection/top:Sap01/top:refId, "SAP_01", /top:Sap01Collection/top:Sap01/top:id, "UPDATE").
```
30. In the **columnName** field, enter "COMMON" or click **Search** to select the column name.
31. In the **needException** field, enter `false()` or click **Search** to select this mode.

Figure 47–28 shows a populated Edit Function – lookupXRef dialog.

Figure 47–28 Edit Function – lookupXRef Dialog: XrefCustApp Use Case

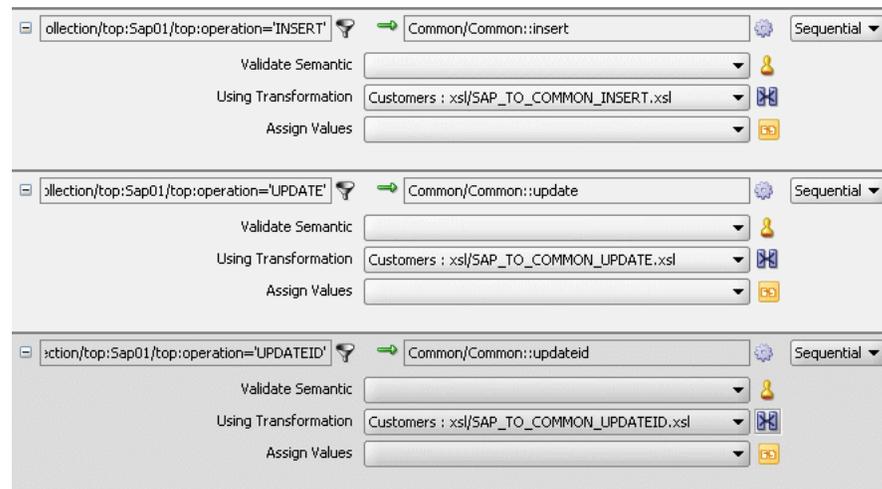


32. Click **OK**.

33. Click **Save All** and close the SAP_TO_COMMON_UPDATEID.xsl window.

The Routing Rules panel would appear as shown in [Figure 47–29](#).

Figure 47–29 Insert, Update, and UpdateID Operations



To create routing rules for delete operation:

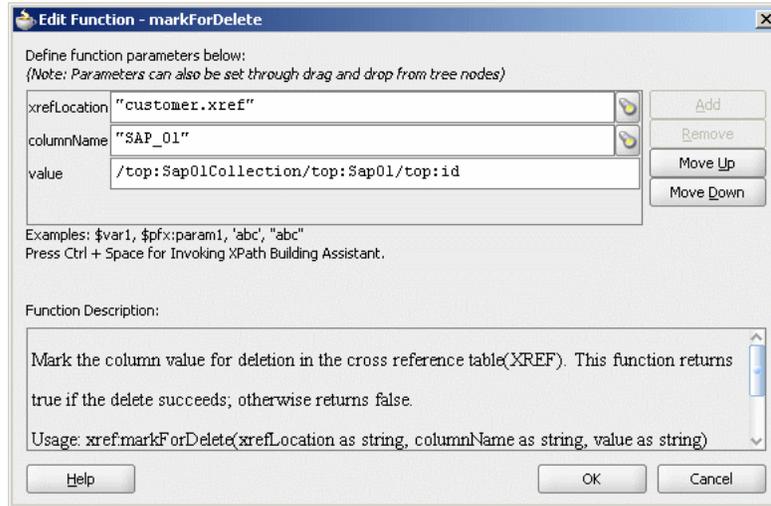
1. In Routing Rules panel, click the **Create a new Routing Rule** icon.
The Target Type dialog is displayed.
2. Select **Service**.
The Target Services dialog is displayed.
3. Navigate to **XrefCustApp, Mediators, Common, Services, Common**.
4. Select **delete** and click **OK**.
5. Click the **Filter** icon.

The Expression Builder dialog is displayed.

6. Enter the following expression in the **Expression** field:
`$in.Sap01Collection/top:Sap01Collection/top:Sap01/top:operation = 'DELETE'`
7. Click **OK**.
8. Click the **Transformation** icon next to the **Transform Using** field.
 The Request Transformation map dialog is displayed.
9. Select **Create New Mapper File** and enter `SAP_TO_COMMON_DELETE.xml`.
10. Click **OK**.
 A `SAP_TO_COMMON_DELETE.xml` tab is displayed.
11. Right-click **<sources>** and select **Add Parameter**.
 The Add Parameter dialog is displayed.
12. In the **Local Name** field, enter `COMMONID`.
13. Select **Set Default Value**.
14. Select **Expression**.
15. In the **XPath Expression** field, enter
`xref:lookupXRef("customer.xref", "SAP_01", /top:Sap01Collection/top:Sap01/top:id, "COMMON", false())`.
16. Click **OK**.
17. Drag and drop the **top:Sap01** source element to the **inp1:Customer** target element.
 The Auto Map Preferences dialog is displayed.
18. Click **OK**.
19. Delete the line connecting **top:id** and **inp1:id**.
20. Connect the **COMMONID** to **inp1:id**.
21. Right-click **inp1:id** and select **Add XSL node** and then **if**.
 A new node `if` is inserted between `inp1:customer` and `inp1:id`.
22. Connect **top:id** to the **if** node.
23. From the **Components** Palette, select **Advanced**.
24. Select **XREF Functions**.
25. Drag and drop the **markForDelete** function from Component Palette to the line connecting **top:id** and **if** node.
26. Double-click the **markForDelete** icon.
 The Edit Function-markForDelete dialog is displayed.
27. Click **Search** to the right of **xrefLocation** field.
 The SCA Resource Lookup dialog is displayed.
28. Select **customer.xref** and click **OK**.
29. In the **columnName** field, enter `"SAP_01"` or click **Search** to select the column name.
30. In the **value** field, enter `/top:Sap01Collection/top:Sap01/top:Id`.

Figure 47–30 shows a populated Edit Function – markForDelete dialog.

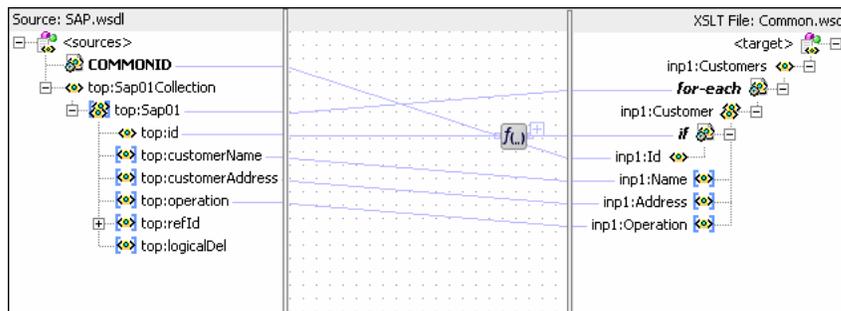
Figure 47–30 Edit Function – markForDelete Dialog: XrefCustApp Use Case



31. Click **OK**.

The SAP_TO_COMMON_DELETE.xsl would appear as shown in Figure 47–31.

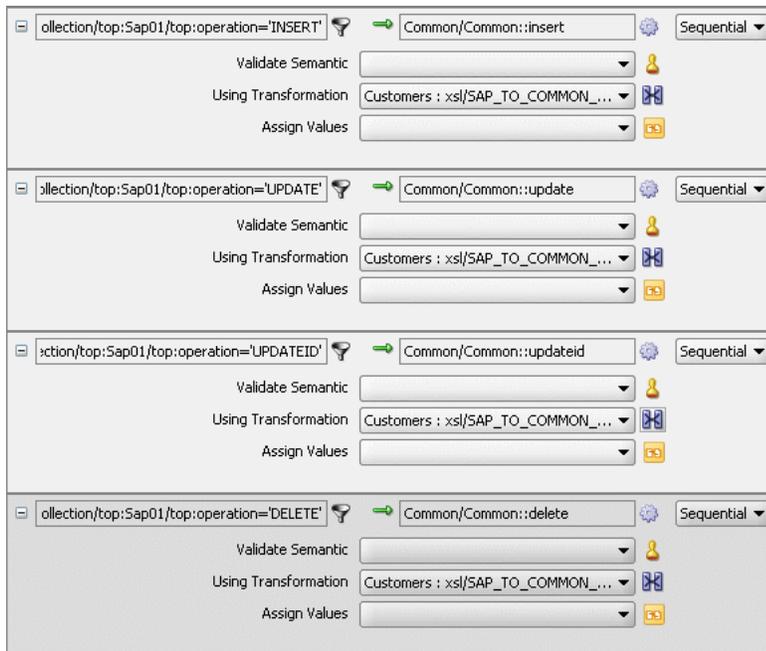
Figure 47–31 SAP_TO_COMMON_DELETE.xsl



32. Click **Save All** and close the SAP_TO_COMMON_DELETE.xsl tab.

The Routing Rules panel would appear as shown in Figure 47–32.

Figure 47–32 Insert, Update, UpdateID, and Delete Operations



47.6.1.9 Task 9: Specifying Routing Rules for Common Mediator

You must specify routing rules for following operations of Common Mediator:

- Insert
- Delete
- Update
- UpdateID

To create routing rules for insert operation:

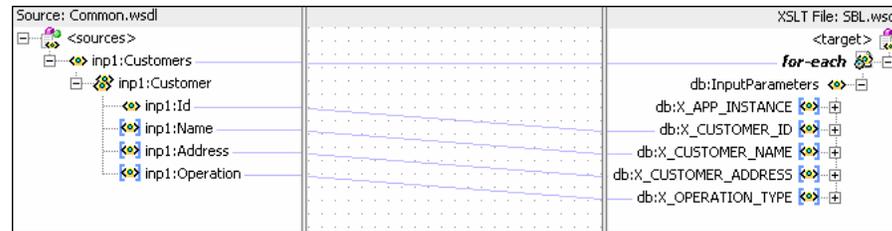
1. Double-click **Common Mediator**.
The Mediator Editor is displayed.
2. In Routing Rules panel, click the **Create a new Routing Rule** icon.
The Target Type dialog is displayed.
3. Select **Service**.
The Target Services dialog is displayed.
4. Navigate to **XrefCustApp, References, SBL**.
5. Select **SBL** and click **OK**.
6. Click the **Transformation** icon next to the **Transform Using** field.
The Request Transformation map dialog is displayed.
7. Select **Create New Mapper File** and enter `COMMON_TO_SBL_INSERT.xml`.
8. Click **OK**.
A `COMMON_TO_SBL_INSERT.xml` tab is displayed.
9. Drag and drop the **inp1:Customers** source element to the **db:InputParameters** target element.

The Auto Map Preferences dialog is displayed.

10. Click **OK**.

The transformation is created as shown in [Figure 47–33](#).

Figure 47–33 *COMMON_TO_SBL_INSERT.xsl Transformation*



11. From the **File** menu, click **Save All** and close the COMMON_TO_SBL_INSERT.xsl window.

12. In the Synchronous Reply panel, click **Browse for target service operations**.

The Target Type dialog is displayed.

13. Select **Service**.

The Target Services dialog is displayed.

14. Navigate to **XrefCustApp, References, Logger**.

15. Select **Write** and click **OK**.

16. Click the **Transformation** icon next to the **Transform Using** field.

The Reply Transformation map dialog is displayed.

17. Select **Create New Mapper File** and enter `SBL_TO_COMMON_INSERT.xsl`.

18. Select **Include Request in the Reply Payload**.

19. Click **OK**.

A SBL_TO_COMMON_INSERT.xsl window is displayed.

20. Connect **inp1:Customers** source element to the **db:X:APP_ID**.

21. Drag and drop the **populateXRefRow** function from Components Palette to the connecting line.

22. Double-click the **populateXRefRow** icon.

The Edit Function-populateXRefRow dialog is displayed.

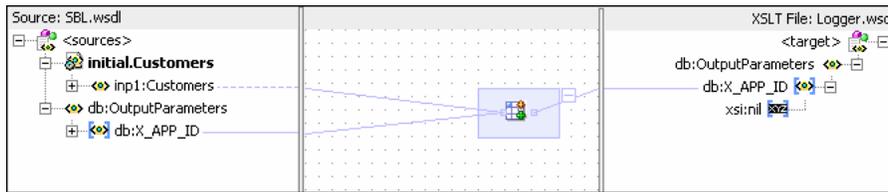
23. Enter the information in the following fields:

- **xrefLocation:** "customer.xref"
- **referenceColumnName:** "Common"
- **referenceValue:** \$initial.Customers/inp1:Customers/inp1:Customer/inp1:Id
- **columnName:** "SBL_78"
- **value:** /db:OutputParameters/db:X_APP_ID
- **mode:** "LINK"

24. Click **OK**.

The SBL_TO_COMMON_INSERT.xml would appear as shown in [Figure 47-34](#).

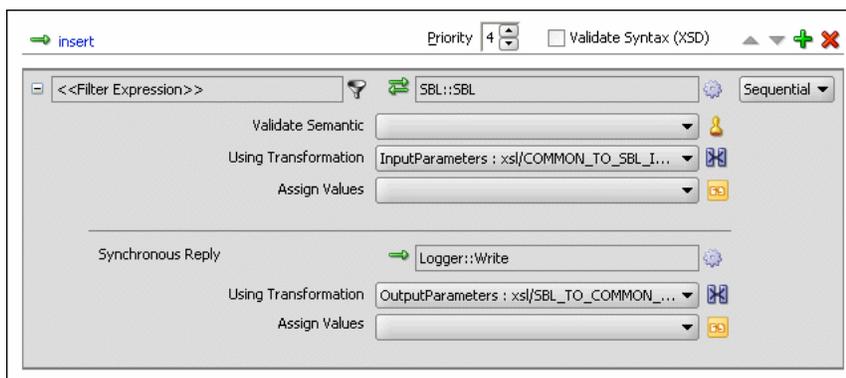
Figure 47-34 SBL_TO_COMMON_INSERT.xml Transformation



25. From the **File** menu, click **Save All** and close the SBL_TO_COMMON_INSERT.xml tab.
26. In the Synchronous Reply panel, click the **Assign Values** icon.
The Assign Values dialog is displayed.
27. Click **Add**.
The Assign Value dialog is displayed.
28. In the **From** section, select **Expression**.
29. Click the **Invoke Expression Builder** icon.
The Expression Builder dialog is displayed.
30. Enter the following expression in the **Expression** field and click **OK**.
`concat('INSERT-', $in.OutputParameters/db:OutputParameters/db:X_APP_ID, '.xml')`
31. In the **To** section, select **Property**.
32. Select **java.fileName** property and click **OK**.
33. Click **OK**.

The insert operation panel would appear as shown in [Figure 47-35](#).

Figure 47-35 Insert Operation with SBL Target Service



34. From the **File** menu, click **Save All**.
35. Repeat the Step 2 through Step 34 to specify another target service EBS and its routing rules.

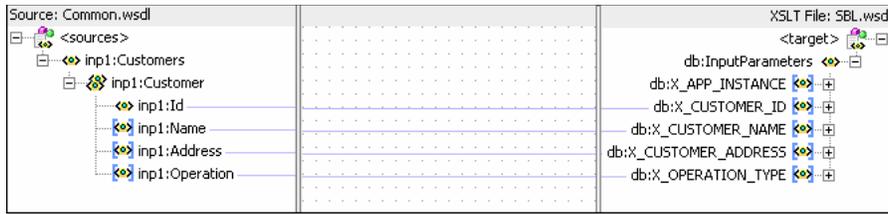
[Figure 47-36](#) shows the insert operation panel with SBL and EBS target service.

Figure 47–36 Insert Operation with SBL and EBS Target Service**To create routing rules for delete operation:**

Perform the following tasks to create the routing rules for delete operation:

1. In Routing Rules panel, click the **Create a new Routing Rule** icon.
The Target Type dialog is displayed.
2. Select **Service**.
The Target Services dialog is displayed.
3. Navigate to **XrefCustApp, References, SBL**.
4. Select **SBL** and click **OK**.
5. Click the **Transformation** icon next to the **Transform Using** field.
The Request Transformation map dialog is displayed.
6. Select **Create New Mapper File** and enter `COMMON_TO_SBL_DELETE.xsl`.
7. Click **OK**.
A `COMMON_TO_SBL_DELETE.xsl` tab is displayed.
8. Drag and drop the `inp1:Customers` source element to the `db:InputParameters` target element.
The Auto Map Preferences dialog is displayed.
9. Click **OK**.
The transformation is created as shown in [Figure 47–37](#).

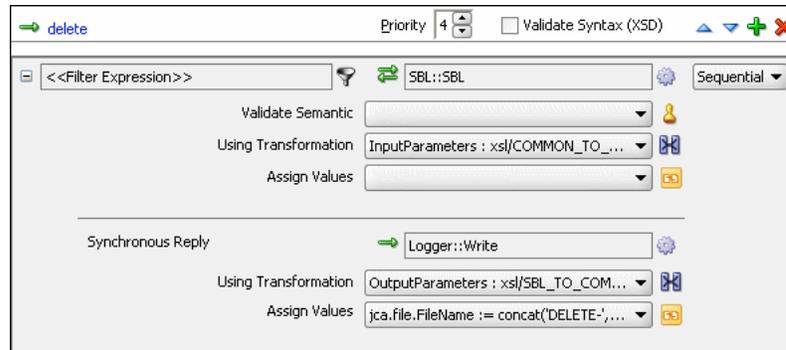
Figure 47-37 COMMON_TO_SBL_DELETE.xsl Transformation



10. Drag and drop the **lookupXRef** function from Components Palette to the line connecting **inp1:id** and **db:XCUSTOMER_ID**.
11. Double-click the **lookupXRef** icon.
The Edit Function: lookupXRef dialog is displayed.
12. Enter the information in the following fields:
 - **xrefLocation**: "customer.xref"
 - **referenceColumnName**: "Common"
 - **referenceValue**: /inp1:Customers/inp1:Customer/inp1:Id
 - **columnName**: "SBL_78"
 - **needException**: false()
13. Click **OK**.
14. From the **File** menu, click **Save All** and close the COMMON_TO_SBL_DELETE.xsl window.
15. In the Synchronous Reply panel, click **Browse for target service operations**.
The Target Type dialog is displayed.
16. Select **Service**.
The Target Services dialog is displayed.
17. Navigate to **XrefCustApp, References, Logger**.
18. Select **Write** and click **OK**.
19. Click the **Transformation** icon next to the **Transform Using** field.
The Reply Transformation map dialog is displayed.
20. Select **Create New Mapper File** and enter **SBL_TO_COMMON_DELETE.xsl**.
21. Click **OK**.
A SBL_TO_COMMON_DELETE.xsl window is displayed.
22. Connect **db:X_APP_ID** source element to the **db:X:APP_ID** target.
23. Drag and drop the **markForDelete** function from Components Palette to the connecting line.
24. Double-click the **markForDelete** icon.
The Edit Function-markForDelete dialog is displayed.
25. Enter the information in the following fields:
 - **xrefLocation**: "customer.xref"
 - **columnName**: "SBL_78"

- **value:**/db:OutputParameters/db:X_APP_ID
- 26. Click **OK**.
- 27. From the **File** menu, click **Save All** and close the SBL_TO_COMMON_DELETE.xml tab.
- 28. In the Synchronous Reply panel, click the **Assign Values** icon.
The Assign Values dialog is displayed.
- 29. Click **Add**.
The Assign Value dialog is displayed.
- 30. In the **From** section, select **Expression**.
- 31. Click the **Invoke Expression Builder** icon.
The Expression Builder dialog is displayed.
- 32. Enter following expression in the **Expression** field and click **OK**.
`concat('DELETE-', $in.OutputParameters/db:OutputParameters/db:X_APP_ID, '.xml')`
- 33. In the **To** section, select **Property**.
- 34. Select **jca.file.FileName** property and click **OK**.
- 35. Click **OK**.
The delete operation panel would appear as shown in [Figure 47–38](#).

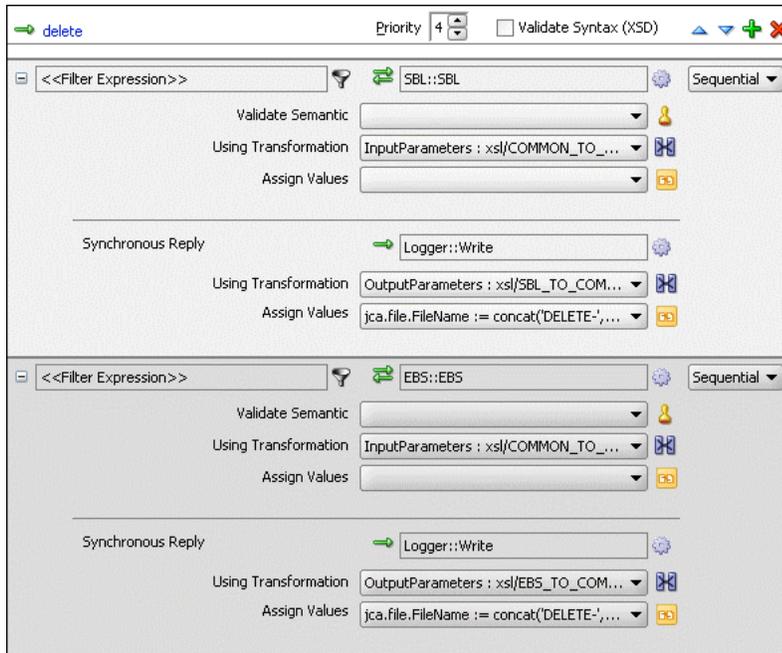
Figure 47–38 Delete Operation with SBL Target Service



- 36. From the **File** menu, click **Save All**.
- 37. Repeat the Step 1 through Step 36 to specify another target service EBS and specify the routing rules.

[Figure 47–39](#) shows the delete operation panel with SBL and EBS target service.

Figure 47–39 Delete Operation with SBL and EBS Target Service



To create routing rules for update operation:

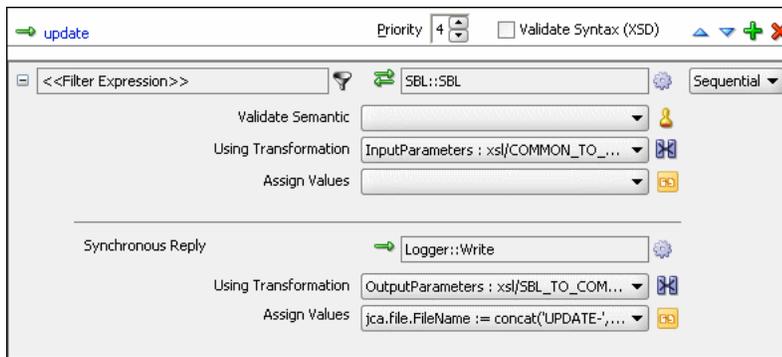
Perform the following tasks to create routing rules for Update operation:

1. In Routing Rules panel, click the **Create a new Routing Rule** icon.
The Target Type dialog is displayed.
2. Select **Service**.
The Target Services dialog is displayed.
3. Navigate to **XrefCustApp, References, SBL**.
4. Select **SBL** and click **OK**.
5. Click the **Transformation** icon next to the **Transform Using** field.
The Request Transformation map dialog is displayed.
6. Select **Create New Mapper File** and enter `COMMON_TO_SBL_UPDATE.xsl`.
7. Click **OK**.
A `COMMON_TO_SBL_UPDATE.xsl` tab is displayed.
8. Drag and drop the **inp1:Customers** source element to the **db:InputParameters** target element.
The Auto Map Preferences dialog is displayed.
9. Click **OK**.
The transformation is created as shown in [Figure 47–37](#).
10. Drag and drop the **lookupXRef** function from Components Palette to the line connecting **inp1:id** and **db:XCUSTOMER_ID**.
11. Double-click the **lookupXRef** icon.
The Edit Function: lookupXRef dialog is displayed.

12. Enter the information in the following fields:
 - **xrefLocation:** "customer.xref"
 - **referenceColumnName:** "Common"
 - **referenceValue:** /inp1:Customers/inp1:Customer/inp1:Id
 - **columnName:**"SBL_78"
 - **needException:**true()
13. Click **OK**.
14. From the **File** menu, click **Save All** and close the COMMON_TO_SBL_UPDATE.xsl window.
15. In the Synchronous Reply panel, click **Browse for target service operations**.
The Target Type dialog is displayed.
16. Select **Service**.
The Target Services dialog is displayed.
17. Navigate to **XrefCustApp, References, Logger**.
18. Select **Write** and click **OK**.
19. Click the **Transformation** icon next to the **Transform Using** field.
The Reply Transformation map dialog is displayed.
20. Select **Create New Mapper File** and enter SBL_TO_COMMON_UPDATE.xsl.
21. Click **OK**.
A SBL_TO_COMMON_UPDATE.xsl window is displayed.
22. Connect **db:X:APP_ID** source element to the **db:X:APP_ID**.
23. From the **File** menu, click **Save All** and close the SBL_TO_COMMON_UPDATE.xsl tab.
24. In the Synchronous Reply panel, click the **Assign Values** icon.
The Assign Values dialog is displayed.
25. Click **Add**.
The Assign Value dialog is displayed.
26. In the **From** section, select **Expression**.
27. Click the **Invoke Expression Builder** icon.
The Expression Builder dialog is displayed.
28. Enter following expression in the **Expression** field and click **OK**.

```
concat('UPDATE-', $in.OutputParameters/db:OutputParameters/db:X_APP_ID, '.xml')
```
29. In the **To** section, select **Property**.
30. Select **jca.file.FileName** property and click **OK**.
31. Click **OK**.
The update operation panel would appear as shown in [Figure 47-40](#).

Figure 47–40 Update Operation with SBL Target Service

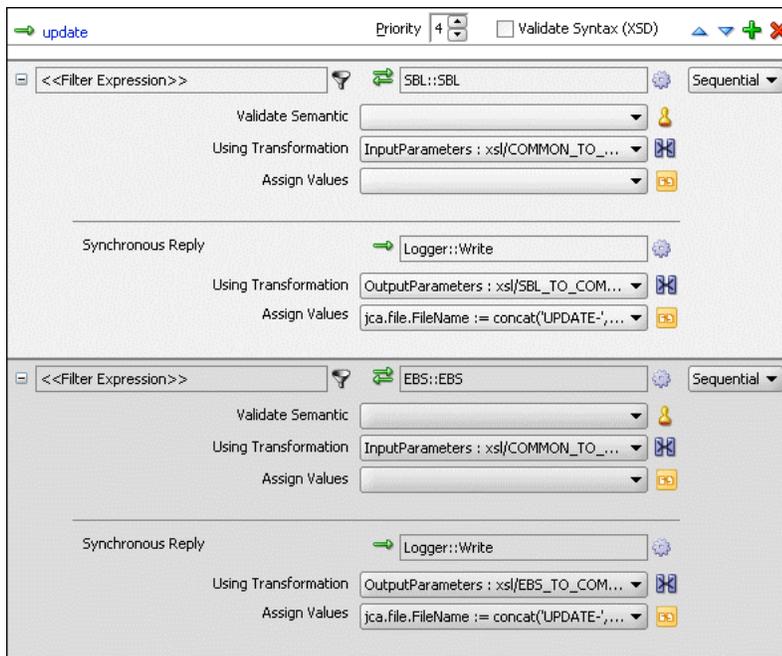


32. From the **File** menu, click **Save All**.

33. Repeat the Step 1 through Step 32 to specify another target service EBS and its routing rules.

Figure 47–41 shows the update operation panel with SBL and EBS target service.

Figure 47–41 Update Operation with SBL and EBS Target Service



To create routing rules for updateID operation:

Perform the following tasks to create routing rules for **UpdateID** operation:

1. In Routing Rules panel, click the **Create a new Routing Rule** icon.
The Target Type dialog is displayed.
2. Select **Service**.
The Target Services dialog is displayed.
3. Navigate to **XrefCustApp, References, SBL**.
4. Select **SBL** and click **OK**.

5. Click the **Transformation** icon next to the **Transform Using** field.
The Request Transformation map dialog is displayed.
6. Select **Create New Mapper File** and enter `COMMON_TO_SBL_UPDATEID.xsl`.
7. Click **OK**.
A `COMMON_TO_SBL_UPDATEID.xsl` tab is displayed.
8. Drag and drop **inp1:Customers** source element to the **db:InputParameters** target element.
The Auto Map Preferences dialog is displayed.
9. Click **OK**.
The transformation is created as shown in [Figure 47–37](#).
10. Drag and drop the **lookupXRef** function from Components Palette to the line connecting **inp1:id** and **db:X_CUSTOMER_ID**.
11. Double-click the **lookupXRef** icon.
The Edit Function: lookupXRef dialog is displayed.
12. Enter the information in the following fields:
 - **xrefLocation**: "customer.xref"
 - **referenceColumnName**: "Common"
 - **referenceValue**: /inp1:Customers/inp1:Customer/inp1:Id
 - **columnName**: "SBL_78"
 - **needException**: false()
13. Click **OK**.
14. From the **File** menu, click **Save All** and close the `COMMON_TO_SBL_UPDATEID.xsl` window.
15. In the Synchronous Reply panel, click **Browse for target service operations**.
The Target Type dialog is displayed.
16. Select **Service**.
The Target Services dialog is displayed.
17. Navigate to **XrefCustApp, References, Logger**.
18. Select **Write** and click **OK**.
19. Click the **Transformation** icon next to the **Transform Using** field.
The Reply Transformation map dialog is displayed.
20. Select **Include Request in the Reply Payload**.
21. Click **OK**.
A `SBL_TO_COMMON_UPDATEID.xsl` window is displayed.
22. Connect **inp1:Customers** source element to the **db:X:APP_ID**.
23. Drag and drop the **populateXRefRow** function from Component Palette to the connecting line.
24. Double-click the **populateXRefRow** icon.

The Edit Function-populateXRefRow dialog is displayed.

25. Enter the information in the following fields:
 - **xrefLocation:** "customer.xref"
 - **referenceColumnName:** "Common"
 - **referenceValue:** \$initial.Customers/inp1:Customers/inp1:Customer/inp1:Id
 - **columnName:**"SBL_78"
 - **value:**/db:OutputParameters/db:X_APP_ID
 - **mode:**"UPDATE"

26. Click **OK**.

27. From the **File** menu, click **Save All** and close the SBL_TO_COMMON_UPDATEID.xsl tab.

28. In the Synchronous Reply panel, click the **Assign Values** icon.

The Assign Values dialog is displayed.

29. Click **Add**.

The Assign Value dialog is displayed.

30. In the **From** section, select **Expression**.

31. Click the **Invoke Expression Builder** icon.

The Expression Builder dialog is displayed.

32. Enter following expression in the **Expression** field and click **OK**.

```
concat('UPDATEID-', $in.OutputParameters/db:OutputParameters/db:X_APP_ID, '.xml')
```

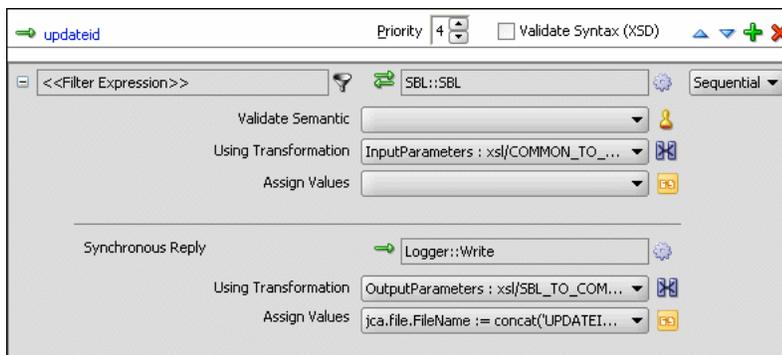
33. In the **To** section, select **Property**.

34. Select **jca.file.FileName** property and click **OK**.

35. Click **OK**.

The updateid operation panel would appear as shown in [Figure 47-42](#).

Figure 47-42 Updateid Operation with SBL Target Service

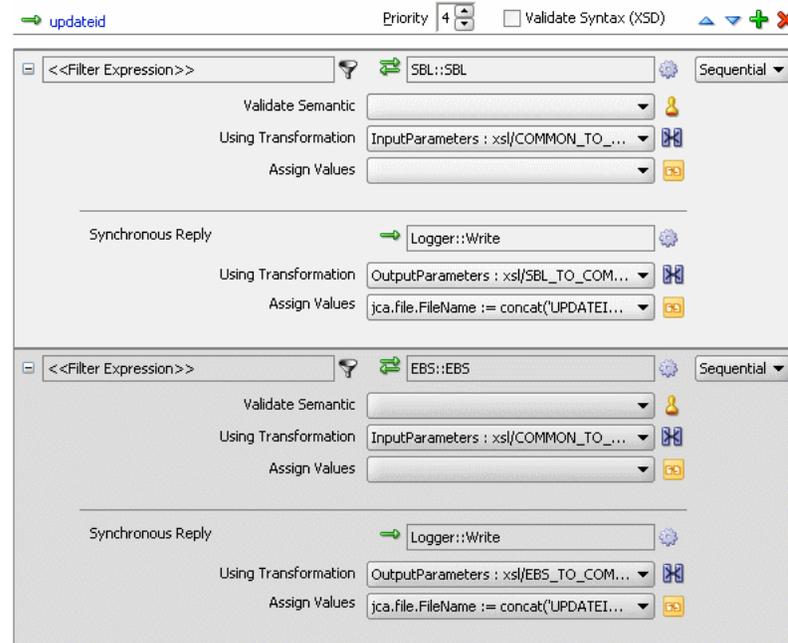


36. From the **File** menu, click **Save All**.

37. Repeat the Step 1 through Step 36 to specify another target service EBS and specify the routing rules.

Figure 47–43 shows the `updateid` operation panel with SBL and EBS target service.

Figure 47–43 Updateid Operation with SBL and EBS Target Service



47.6.1.10 Task 10: Configuring Oracle Application Server Connection

An Oracle Application Server connection is required for deploying your SOA composite application. For information on creating Oracle Application Server connection, refer to *Oracle Fusion Middleware User's Guide for Technology Adapters*.

47.6.1.11 Task 11: Deploying the Composite Application

Deploying the `XrefCustApp` composite application to Oracle Application Server consists of following steps:

- Creating an Application Deployment Profile
- Deploying the Application to Oracle Application Server

For detailed information about these steps, see [Section 43.2, "Deploying a Single SOA Composite in Oracle JDeveloper"](#).

47.6.2 Running and Monitoring the XrefCustApp Application

After deploying the `XrefCustApp` application, you can run it by using any command from the `insert_sap_record.sql` file present in the `XrefCustApp/sql` folder. On successful completion, the records are inserted or updated in EBS and SBL tables and the `Logger` reference writes the output to the `output.xml` file.

For monitoring the running instance, you can use the Oracle Enterprise Manager Console at the following URL:

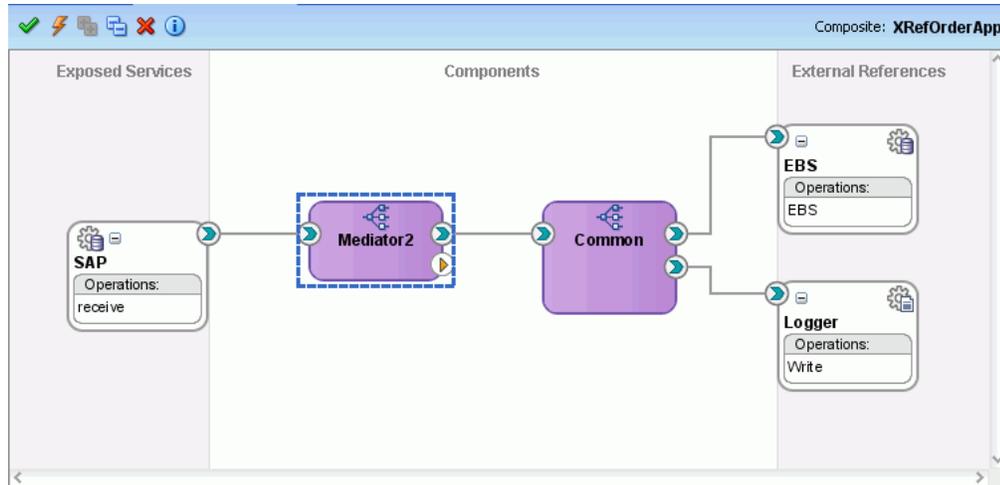
```
http://hostname:portnumber/em
```

where `hostname` is the host on which you installed the Oracle SOA Suite infrastructure and `portnumber` is the port running the service.

47.7 Creating and Running Cross Reference for 1M Functions

The cross reference use case implements an integration scenario between two end-systems Oracle EBS and SAP instances. In this use case, the order passes from SAP to EBS. SAP represents the orders with a unique ID, whereas EBS splits the order into two order ID1 and ID2. This scenario is created using Database Adapters. When you poll the SAP table for updated or created records, a SAP instance is created. In EBS, the instance is simulated by a procedure and the table is populated. [Figure 47–44](#) provides an overview of this use case.

Figure 47–44 *XrefOrderApp Use Case in SOA Composite Editor*



For downloading the sample files mentioned in this section, visit the following URL:
http://www.oracle.com/technology/sample_code/products/mediator

47.7.1 Step-By-Step Instructions for Creating the Use Case

This section provides the design-time tasks for creating, building, and deploying your SOA Composite application. These tasks should be performed in the order in which they are presented.

- [Section 47.7.1.1, "Task 1: Configuring Oracle Database and Database Adapter"](#)
- [Section 47.7.1.2, "Task 2: Creating an Oracle JDeveloper Application and Project"](#)
- [Section 47.7.1.3, "Task 3: Creating a Cross Reference"](#)
- [Section 47.7.1.4, "Task 4: Creating a Database Adapter Service"](#)
- [Section 47.7.1.5, "Task 5: Creating EBS External Reference"](#)
- [Section 47.7.1.6, "Task 6: Creating Logger External Reference"](#)
- [Section 47.7.1.7, "Task 7: Creating Mediator Components"](#)
- [Section 47.7.1.8, "Task 8: Specifying Routing Rules for Mediator Component"](#)
- [Section 47.7.1.9, "Task 9: Specifying Routing Rules for Common Mediator"](#)
- [Section 47.7.1.10, "Task 10: Configuring Oracle Application Server Connection"](#)
- [Section 47.7.1.11, "Task 11: Deploying the Composite Application"](#)

47.7.1.1 Task 1: Configuring Oracle Database and Database Adapter

To configure Oracle Database and the Database adapter

1. You need SCOTT database account with password TIGER for this use case. You must ensure that the SCOTT account is unlocked.

You can log in as SYSDBA and then run the `setup_user.sql` script available in the `XrefOrderApp1M/sql` folder to unlock the account.
2. Run the `create_schema.sql` script available in the `XrefOrderApp1M/sql` folder to create the tables required for this use case.
3. Run the `create_app_procedure.sql` script available in the `XrefOrderApp1M/sql` folder to create a procedure that simulates the various applications participating in this integration.
4. Run the `createschema_xref_oracle.sql` script available in the `OH/rcu/integration/soainfra/sql/xref/` folder to create a Cross Reference table to store runtime Cross Reference data.
5. Copy the `ra.xml` and `weblogic-ra.xml` files from `$BEAHOME/META-INF` to the newly created directory called `META-INF` on your computer.
6. Edit the `weblogic-ra.xml` file, available in the `$BEAHOME/src/oracle/tip/adapter/db/test/deploy/weblogic/META-INF` folder from the ADE label that you are using for your SOA application, as follows:

- Modify the property to `xADataSourceName` as follows:

```
<property>
  <name>xADataSourceName</name>
  <value>jdbc/DBConnection1</value>
</property>
```

- Modify the `jndi-name` as follows:

```
<jndi-name> eis/DB/DBConnection1</jndi-name>
```

This sample uses `eis/DB/DBConnection1` to poll SAP table for new messages and to connect to the procedure that simulates Oracle EBS and Siebel instances.

7. Package the `ra.xml` and `weblogic-ra.xml` files as a RAR file and deploy the RAR file by using the Weblogic console.
8. Create a data source using the Weblogic console with the following values:
 - **jndi-name**=`jdbc/DBConnection1`
 - **user**=`scott`
 - **password**=`tiger`
 - **url**=`jdbc:oracle:thin:@host:port:service`
 - **connection-factory**
factory-class=`oracle.jdbc.pool.OracleDataSource`
9. Create a data source using the Weblogic console with the following values:
 - **jndi-name**=`jdbc/xref`
 - **user**=`scott`
 - **password**=`tiger`

- `url=jdbc:oracle:thin:@host:port:service`
- **connection-factory**
`factory-class=oracle.jdbc.pool.OracleDataSource`

47.7.1.2 Task 2: Creating an Oracle JDeveloper Application and Project

To create an application and a project:

1. In Oracle JDeveloper, click **File** and select **New**.
The New Gallery dialog appears.
2. In the New Gallery, expand the **General** node, and select the **Applications** category.
3. In the **Items** list, select **SOA Application** and click **OK**.
The Create SOA Application wizard appears.
4. In the **Application Name** field, enter `XRefOrderApp`, and then click **Next**.
The **Name your project** screen appears.
5. In the **Project Name** field, enter `XRefOrderApp` and click **Next**.
The **Configure SOA Settings** screen appears.
6. In the Composite Template list, select **Empty Composite** and then click **Finish**.
The Applications Navigator of Oracle JDeveloper is updated with the new application and project and the Design tab contains, a blank palette.
7. From the **File** menu, click **Save All**.

47.7.1.3 Task 3: Creating a Cross Reference

After creating an application and a project for the use case, you must create a cross reference table.

To create a cross reference table:

1. In the Application Navigator, right-click the `XRefOrderApp` project and select **New**.
2. In the New Gallery dialog, expand the **SOA Tier** node, and then select the **Transformations** category.
3. In the Items list, select **Cross Reference(XREF)** and click **OK**.
The Create Cross Reference(XREF) File dialog is displayed.
4. In the **File Name** field, enter `order.xref`.
5. In the **End System** fields, enter `SAP_05` and `EBS_i75`.
6. Click **OK**.
The Cross Reference Editor is displayed.
7. Click **Add**.
A new row is added.
8. Enter **COMMON** as End System name.
The Cross Reference Editor would appear as shown in [Figure 47-45](#).

Figure 47–45 Customer Cross Reference

The screenshot shows a dialog box titled "Cross Reference(XREF)". It has a "Name:" field with the text "Order" and a "Description:" field which is empty. Below these is a section titled "End Systems:" with a "+" and "-" icon. Underneath, there is a list of system names: "SAP_05", "EBS_I75", and "COMMON".

9. From the **File** menu, click **Save All** and close the Cross Reference Editor.

47.7.1.4 Task 4: Creating a Database Adapter Service

To create a Database adapter service:

1. In the Component Palette, select **SOA**.
2. Select **Database Adapter** and drag it to the Exposed Services design area.
The Adapter Configuration wizard Welcome page is displayed.
3. Click **Next**.
The Service Name page is displayed.
4. In the **Service Name** field, enter **SAP**.
5. Click **Next**.
The Service Connection page is displayed.
6. In the **Connection** field, select **DBConnection1**.
7. In the **JNDI Name** field, enter `eis/DB/DBConnection1`.
8. Click **Next**.
The Operation Type page is displayed.
9. Select **Poll for New or Changed Records in a Table** and click **Next**.
The Select Table page is displayed.
10. Click **Import Tables**.
The Import Tables dialog is displayed.
11. Select **Scott** from Schema.
12. In the **Name Filter** field, enter `%SAP%` and click **Query**.
The **Available** field is populated with `SAP_05` table name.
13. Double-click **SAP_05**.
The **selected** field is populated with `SAP_05`.
14. Click **OK**.

The Select Table page now contains the SAP_05 table.

15. Select **SAP_05** and click **Next**.

The Define Primary Key page is displayed.

16. Select **ID** as primary key and click **Next**.

The Relationships page is displayed.

17. Click **Next**.

The Attribute Filtering page is displayed.

18. Click **Next**.

The After Read page is displayed.

19. Select **Update a Field in the [SAP_05] Table (Logical Delete)** and click **Next**.

The Logical Delete page is displayed.

20. In the **Logical Delete** field, select **LOGICAL_DEL**.

21. In the **Read Value** field, enter **Y**.

22. In the **Unread Value** field, enter **N**.

[Figure 47–14](#) shows the Logical Delete page of the Adapter Configuration Wizard.

23. Click **Next**.

The Polling Options page is displayed.

24. Click **Next**.

The Define Selection Criteria page is displayed.

25. Click **Next**.

The Advanced Options page is displayed.

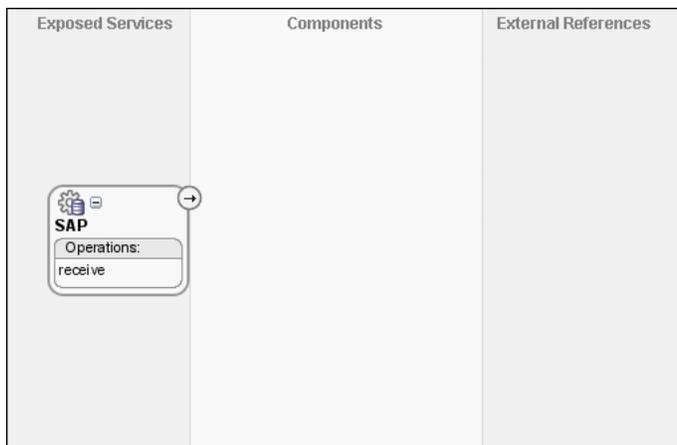
26. Click **Next**.

The Finish page is displayed.

27. Click **Finish**.

A Database adapter service **SAP** is created, as shown in [Figure 47–46](#).

Figure 47–46 SAP Database Adapter Service in SOA Composite Editor



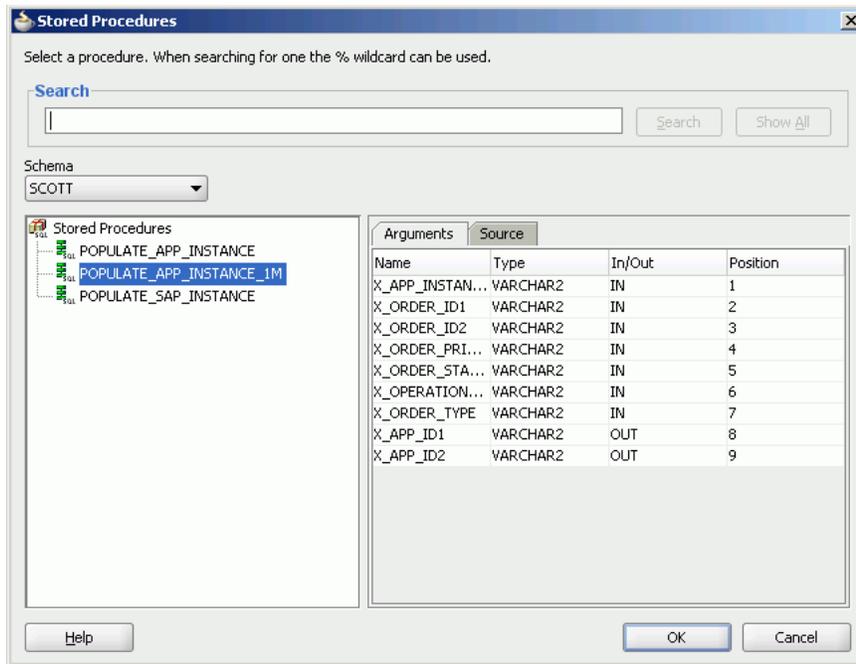
28. From the **File** menu, click **Save All**.

47.7.1.5 Task 5: Creating EBS External Reference

To create external references named **EBS**:

1. In the Component Palette, select **SOA**.
2. Select **Database Adapter** and drag it to the External References design area.
The Adapter Configuration wizard Welcome page is displayed.
3. Click **Next**.
The Service Name page is displayed.
4. In the **Service Name** field, enter **EBS**.
5. Click **Next**.
The Service Connection page is displayed.
6. In the **Connection** field, select **DBConnection1**.
7. In the **JNDI Name** field, enter `eis/DB/DBConnection1`.
8. Click **Next**.
The Operation Type page is displayed.
9. Select **Call a Stored Procedure or Function** and click **Next**.
The Specify Stored Procedure page is displayed.
10. Select **Scott** from Schema.
11. Click **Browse**.
The Stored Procedures dialog is displayed.
12. Select **POPULATE_APP_INSTANCE_IM** as shown in [Figure 47-47](#).

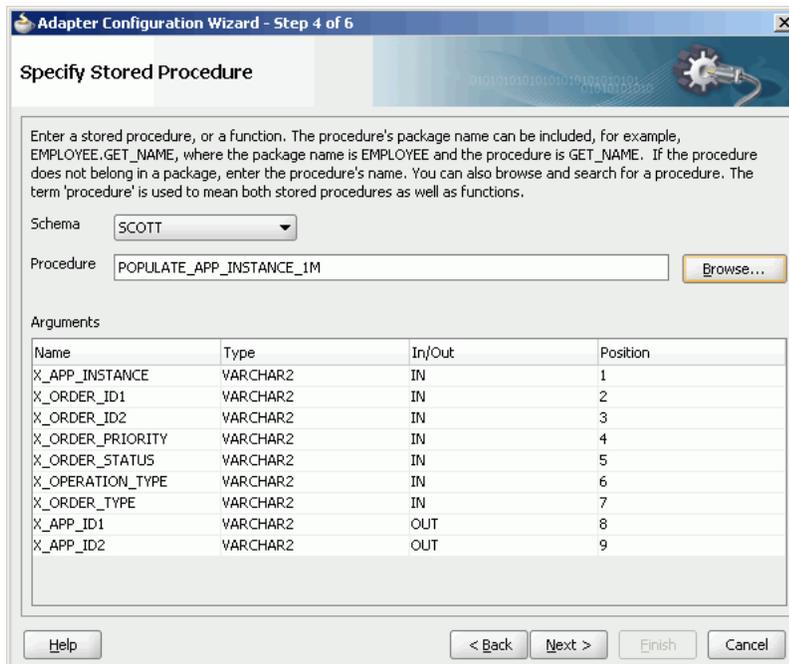
Figure 47–47 Stored Procedure Dialog



13. Click OK.

The Specify Stored Procedure page appears as shown in [Figure 47–48](#).

Figure 47–48 Specify Stored Procedure Page of Adapter Configuration Wizard



14. Click Next.

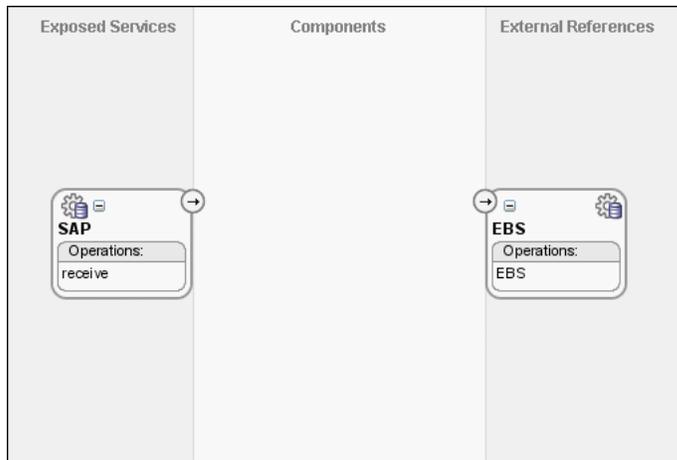
The Advanced Options page is displayed.

15. Click Next.

16. The Finish page is displayed.
17. Click **Finish**.

Figure 47–49 shows the EBS reference in SOA Composite Editor.

Figure 47–49 EBS Reference in SOA Composite Editor



18. From the **File** menu, click **Save All**.

47.7.1.6 Task 6: Creating Logger External Reference

To create a file adapter reference:

1. From the Component Palette, select **SOA**.
2. Select **File Adapter** and drag it to the External References design area.
The Adapter Configuration wizard Welcome page is displayed.
3. Click **Next**.
The Service Name page is displayed.
4. In the **Service Name** field, enter **Logger**.
5. Click **Next**.
The Adapter Interface page is displayed.
6. Click **Define from operation and schema (specified later)**.
The Operation page is displayed.
7. In the **Operation Type** field, select **Write File**.
8. Click **Next**.
The File Configuration page is displayed.
9. In the **Directory for Outgoing Files (physical path)** field, enter the name of the directory where you want to write the files.
10. In the **File Naming Convention** field, enter `output.xml` and click **Next**.
The Messages page is displayed.
11. Click **Search**.

The Type Chooser dialog is displayed.

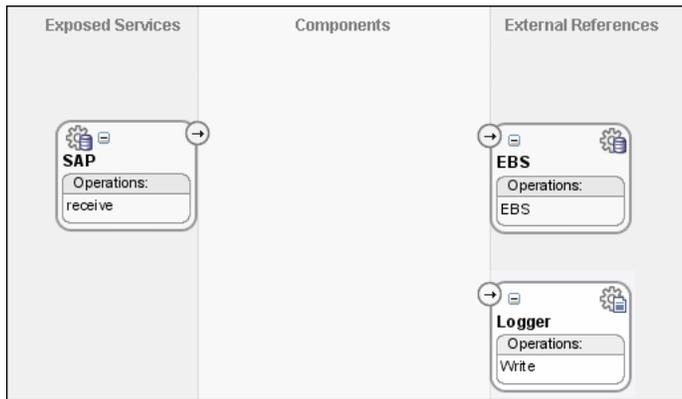
12. Navigate to **Type Explorer, Project Schema Files, SCOTT_POPULATE_APP_INSTANCE_1M.xsd** and then select **OutputParameters**.
13. Click **OK**.
14. Click **Next**.

The Finish page is displayed.

15. Click **Finish**.

[Figure 47-50](#) shows the `Logger` reference in the SOA Composite Editor.

Figure 47-50 *Logger Reference in SOA Composite Editor*



16. From the **File** menu, click **Save All**.

47.7.1.7 Task 7: Creating Mediator Components

To create a Mediator component:

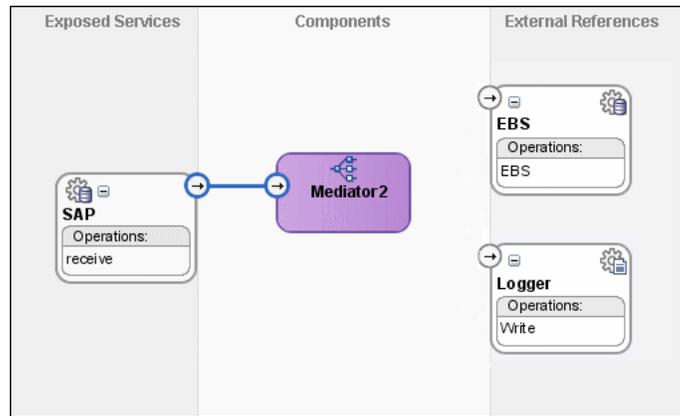
1. Drag and drop a Mediator from Components Palette to the Components design area.

The Create Mediator dialog is displayed.

2. Select **Define Interface Later** from Template.
3. Click **OK**.

A Mediator with name `Mediator2` is created.

4. Connect the **SAP** service to the **Mediator2** as shown in [Figure 47-51](#).

Figure 47–51 SAP Service Connected to Mediator2

5. Click **Save All**.
6. Drag and drop another Mediator from Components Palette to the Components design area.
The Create Mediator dialog is displayed.
7. Select **Interface Definition From WSDL** from Template.
8. Deselect **Create Composite Service with SOAP Bindings**.
9. Click **Find Existing WSDLs** to the right of the **WSDL File** field.
10. Navigate to and then select the `Common.wsdl` file. The `Common.wsdl` file is available in the `Samples` folder.
11. Click **OK**.
12. Click **OK**.

A Mediator with name `Common` is created.

47.7.1.8 Task 8: Specifying Routing Rules for Mediator Component

You must specify routing rules for following operations:

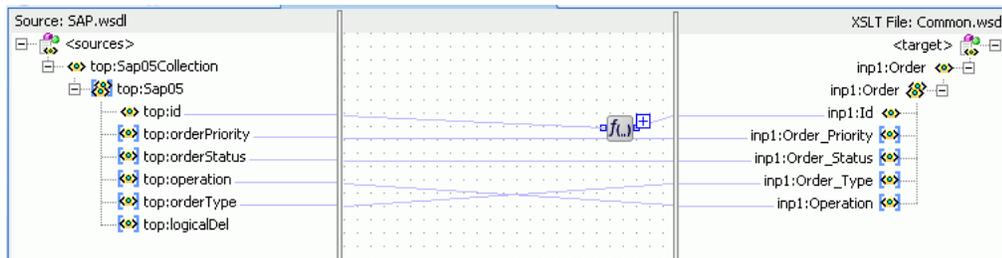
- Insert
- Update

To create routing rules for insert operation:

1. Double-click `Mediator2` Mediator.
The Mediator Editor is displayed.
2. In Routing Rules panel, click the **Create a new Routing Rule** icon.
The Target Type dialog is displayed.
3. Select **Service**.
The Target Services dialog is displayed.
4. Navigate to `XrefOrderApp, Mediators, Common, Services, Common`.
5. Select **Insert** and click **OK**.
6. Click the **Filter** icon.
The Expression Builder dialog is displayed.

7. Enter the following expression in the **Expression** field:
`$in.Sap05Collection/top:Sap05Collection/top:Sap05/top:operation='INSERT'`
8. Click **OK**.
9. Click the **Transformation** icon next to the **Using Transformation** field.
 The Request Transformation map dialog is displayed.
10. Select **Create New Mapper File** and enter `SAP_TO_COMMON_INSERT.xsl`.
11. Click **OK**.
 A `SAP_TO_COMMON_INSERT.xsl` tab is displayed.
12. Drag and drop **top:SAP05** source element to the **inp1:Order** target element.
 The Auto Map Preferences dialog is displayed.
13. From the **During Auto Map** options, deselect **Match Elements Considering their Ancestor Names**.
14. Click **OK**.
 The transformation is created as shown in [Figure 47-52](#).

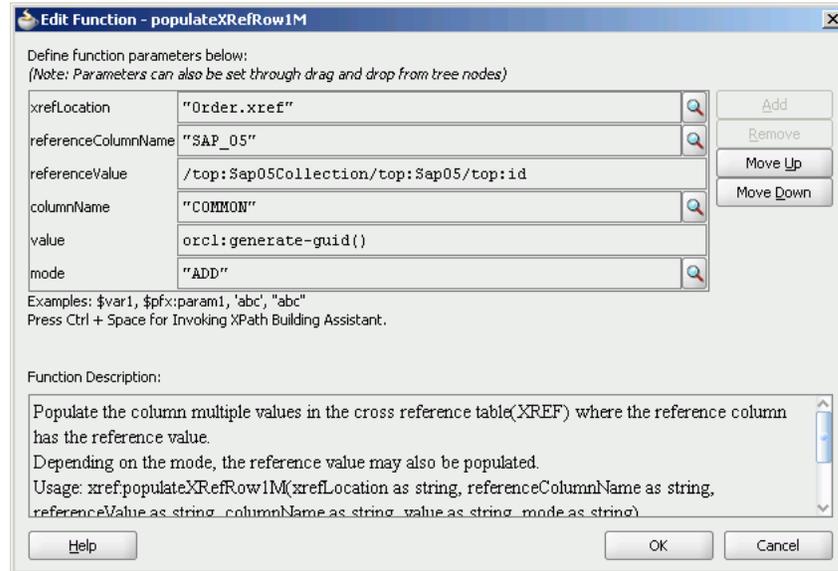
Figure 47-52 `SAP_TO_COMMON_INSERT.xsl` Transformation



15. From the Components Palette, select **Advanced**.
16. Select **XREF Functions**.
17. Drag and drop the **populateXRefRow1M** function from Components Palette to the line connecting **top:id** and **inp1:id** elements.
18. Double-click the **populateXRefRow1M** icon.
 The Edit Function-populateXRefRow dialog is displayed.
19. Click **Search** to the right of **xrefLocation** field.
 The SCA Resource Lookup dialog is displayed.
20. Select **Order.xref** and click **OK**.
21. In the **referenceColumnName** field, enter "SAP_05" or click **Search** to select the column name.
22. In the **referenceValue** column, enter
`/top:Sap05Collection/top:Sap05/top:id`.
23. In the **columnName** field, enter "Common" or click **Search** to select the column name.
24. In the **value** field, enter `orcl:generate-guid()`.
25. In the **mode** field, enter "Add" or click **Search** to select this mode.

Figure 47–53 shows populated Edit Function – populateXRefRow1M dialog.

Figure 47–53 Edit Function – populateXRefRow1M Dialog: XrefOrderApp Use Case

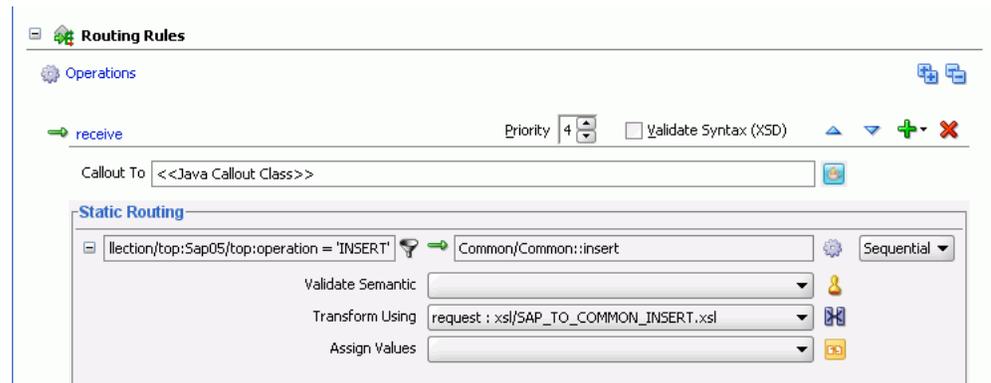


26. Click **OK**.

27. From the **File** menu, click **Save All** and close the `SAP_TO_COMMON_INSERT.xml` tab.

The Routing Rules panel would appear as shown in Figure 47–54.

Figure 47–54 Routing Rules Panel with Insert Operation



To create routing rules for update operation:

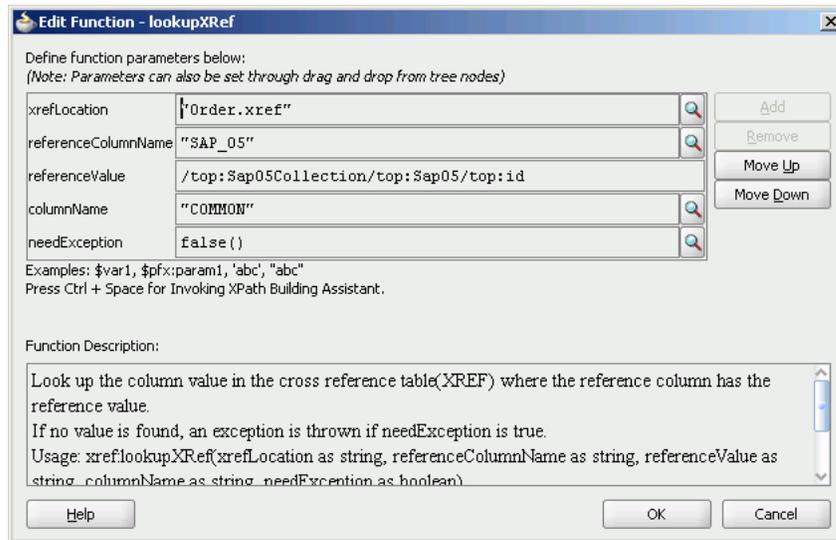
Perform the following tasks to create routing rules for Update operation:

1. In Routing Rules panel, click the **Create a new Routing Rule** icon.
The Target Type dialog is displayed.
2. Select **Service**.
The Target Services dialog is displayed.
3. Navigate to **XrefOrderApp, Mediators, Common, Services, Common**.
4. Select **Update** and click **OK**.

5. Click the **Filter** icon.
The Expression Builder dialog is displayed.
6. Enter the following expression in the **Expression** field:
`$in.Sap05Collection/top:Sap05Collection/top:Sap05/top:operation='UPDATE'`
7. Click **OK**.
8. Click the **Transformation** icon next to the **Transform Using** field.
The Request Transformation map dialog is displayed.
9. Select **Create New Mapper File** and enter `SAP_TO_COMMON_UPDATE.xml`.
10. Click **OK**.
A `SAP_TO_COMMON_UPDATE.xml` tab is displayed.
11. Drag and drop the **top:Sap05** source element to the **inp1:Order** target element.
The Auto Map Preferences dialog is displayed.
12. Click **OK**.
13. From the **Components Palette**, select **Advanced**.
14. Select **XREF Functions**.
15. Drag and drop the **lookupXRef** function from Components Palette to the line connecting **top:id** and **inp1:id** elements.
16. Double-click the **lookupXRef** icon.
The Edit Function-lookupXRef dialog is displayed.
17. Click **Search** to the right of **xrefLocation** field.
The SCA Resource Lookup dialog is displayed.
18. Select **customer.xref** and click **OK**.
19. In the **referenceColumnName** field, enter "SAP_05" or click **Search** to select the column name.
20. In the **referenceValue** column, enter
`/top:Sap05Collection/top:Sap05/top:id`.
21. In the **columnName** field, enter "COMMON" or click **Search** to select the column name.
22. In the **needException** field, enter `true()` or click **Search** to select this mode.

[Figure 47-55](#) shows populated Edit Function – looupXRef dialog.

Figure 47–55 Edit Function – looupXRef Dialog: XRefOrderApp Use Case

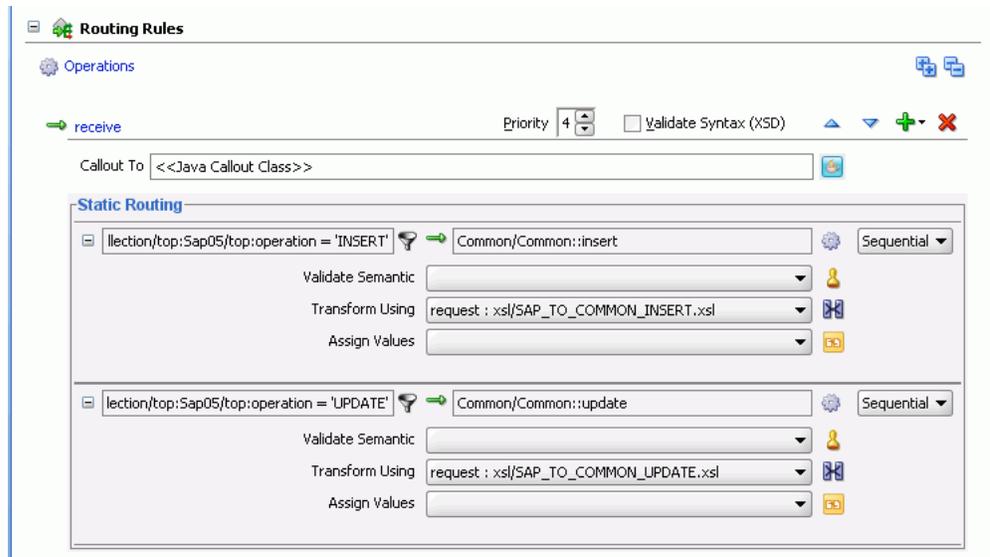


23. Click **OK**.

24. From the **File** menu, click **Save All** and close the SAP_TO_COMMON_UPDATE.xsl tab.

The Routing Rules panel would appear as shown in [Figure 47–56](#).

Figure 47–56 Insert Operation and Update Operation



47.7.1.9 Task 9: Specifying Routing Rules for Common Mediator

You must specify routing rules for following operations of Common Mediator:

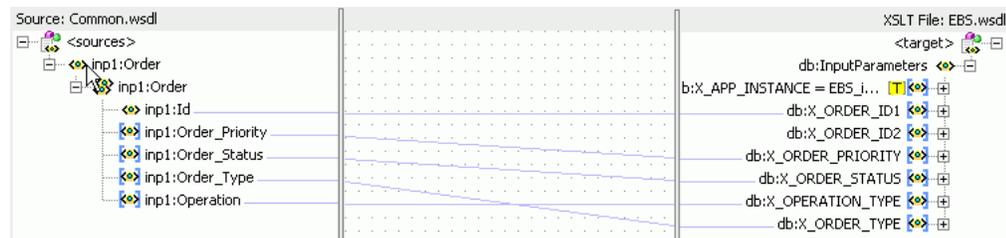
- Insert
- Update

To create routing rules for insert operation:

1. Double-click **Common Mediator**.

- The Mediator Editor is displayed.
2. In Routing Rules panel, click the **Create a new Routing Rule** icon.
The Target Type dialog is displayed.
 3. Select **Service**.
The Target Services dialog is displayed.
 4. Navigate to **XrefOrderApp, References, EBS**.
 5. Select **EBS** and click **OK**.
 6. Click the **Transformation** icon next to the **Transform Using** field.
The Request Transformation map dialog is displayed.
 7. Select **Create New Mapper File** and enter `COMMON_TO_EBS_INSERT.xsl`.
 8. Click **OK**.
A `COMMON_TO_EBS_INSERT.xsl` tab is displayed.
 9. Drag and drop **inp1:Order** source element to the **db:InputParameters** target element.
The Auto Map Preferences dialog is displayed.
 10. Set the value of the `db:X_APP_INSTANCE` node on the right side to `EBS_i75`.
Click **OK**.
The transformation is created as shown in [Figure 47–57](#).

Figure 47–57 *COMMON_TO_EBS_INSERT.xsl Transformation*



11. From the **File** menu, click **Save All** and close the `COMMON_TO_EBS_INSERT.xsl` window.
12. In the Synchronous Reply panel, click **Browse for target service operations**.
The Target Type dialog is displayed.
13. Select **Service**.
The Target Services dialog is displayed.
14. Navigate to **XrefOrderApp, References, Logger**.
15. Select **Write** and click **OK**.
16. Click the **Transformation** icon next to the **Transform Using** field.
The Reply Transformation map dialog is displayed.
17. Select **Create New Mapper File** and enter `EBS_TO_COMMON_INSERT.xsl`.
18. Select **Include Request in the Reply Payload**.

19. Click **OK**.

AN EBS_TO_COMMON_INSERT.xsl window is displayed.

20. Connect **inp1:Order** source element to the **db:X:APP_ID**.

21. Drag and drop the **populateXRefRow** function from Components Palette to the connecting line.

22. Double-click the **populateXRefRow** icon.

The Edit Function-populateXRefRow dialog is displayed.

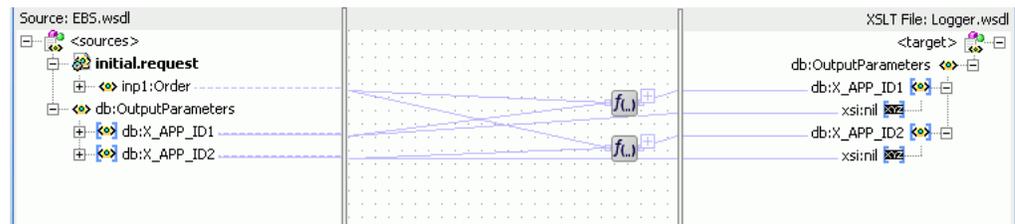
23. Enter the information in the following fields:

- **xrefLocation:** "order.xref"
- **referenceColumnName:** "Common"
- **referenceValue:** \$initial.Customers/inp1:Customers/inp1:Order/inp1:Id
- **columnName:**"EBS_75"
- **value:**/db:OutputParameters/db:X_APP_ID
- **mode:**"LINK"

24. Click **OK**.

The EBS_TO_COMMON_INSERT.xsl would appear as shown in [Figure 47–58](#).

Figure 47–58 EBS_TO_COMMON_INSERT.xsl Transformation



25. From the **File** menu, click **Save All** and close the EBS_TO_COMMON_INSERT.xsl tab.

26. In the Synchronous Reply panel, click the **Assign Values** icon.

The Assign Values dialog is displayed.

27. Click **Add**.

The Assign Value dialog is displayed.

28. In the **From** section, select **Expression**.

29. Click the **Invoke Expression Builder** icon.

The Expression Builder dialog is displayed.

30. Enter the following expression in the **Expression** field and click **OK**.

```
concat('INSERT-', $in.OutputParameters/db:OutputParameters/db:X_APP_ID, '.xml')
```

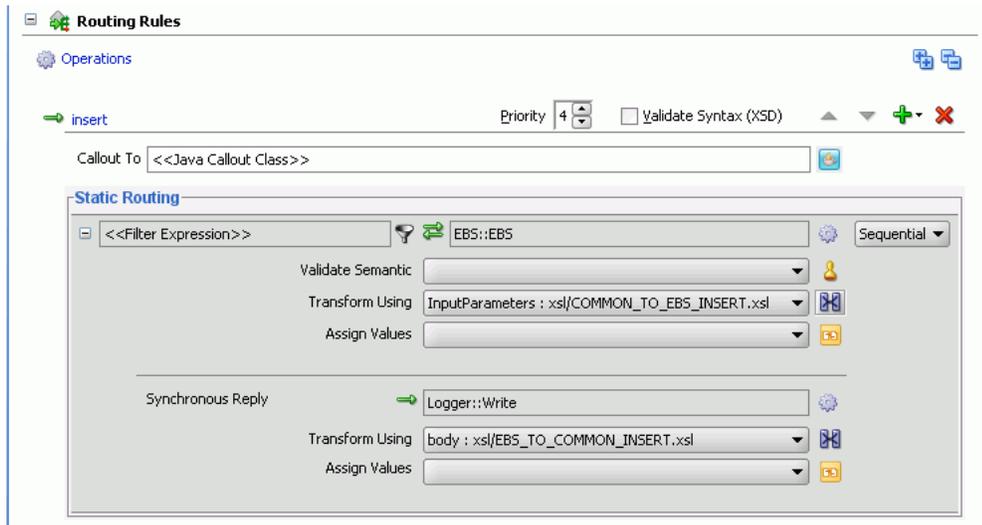
31. In the **To** section, select **Property**.

32. Select **jca.file.FileName** property and click **OK**.

33. Click **OK**.

The insert operation panel would appear as shown in [Figure 47–59](#).

Figure 47–59 Insert Operation with EBS Target Service



34. From the **File** menu, click **Save All**.

To create routing rules for update operation:

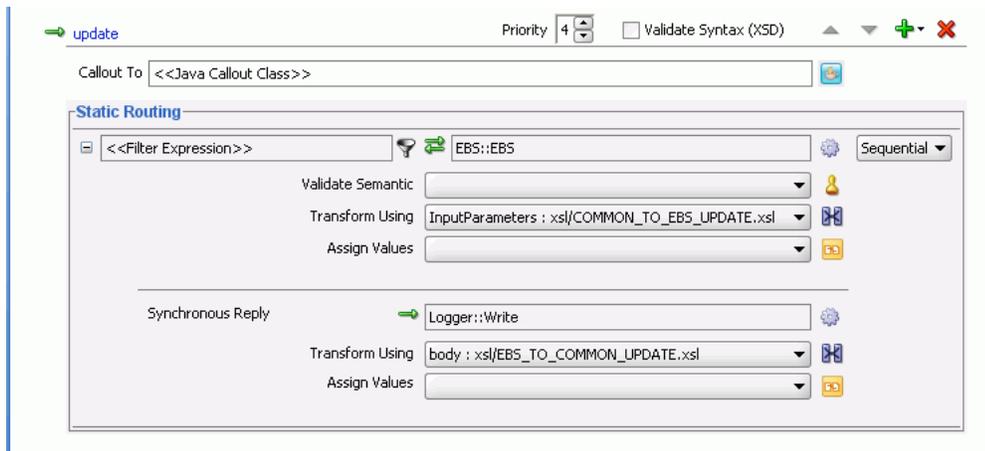
Perform the following tasks to create routing rules for Update operation:

1. In Routing Rules panel, click the **Create a new Routing Rule** icon.
The Target Type dialog is displayed.
2. Select **Service**.
The Target Services dialog is displayed.
3. Navigate to **XrefOrderApp, References, EBS**.
4. Select **EBS** and click **OK**.
5. Click the **Transformation** icon next to the **Transform Using** field.
The Request Transformation map dialog is displayed.
6. Select **Create New Mapper File** and enter `COMMON_TO_EBS_UPDATE.xsl`.
7. Click **OK**.
A `COMMON_TO_EBS_UPDATE.xsl` tab is displayed.
8. Drag and drop **inp1:Orders** source element to the **db:InputParameters** target element.
The Auto Map Preferences dialog is displayed.
9. Click **OK**.
The transformation is created as shown in [Figure 47–37](#).
10. Drag and drop the **lookupXRef** function from Components Palette to the line connecting **inp1:id** and **db:X_APP_ID**.
11. Double-click the **lookupXRef** icon.
The Edit Function: lookupXRef dialog is displayed.

12. Enter the information in the following fields:
 - **xrefLocation:** "order.xref"
 - **referenceColumnName:** "Common"
 - **referenceValue:** /inp1:Customers/inp1:Order/inp1:Id
 - **columnName:**"EBS_i75"
 - **needException:**true()
13. Click **OK**.
14. From the **File** menu, click **Save All** and close the COMMON_TO_EBS_UPDATE.xsl window.
15. In the Synchronous Reply panel, click **Browse for target service operations**.
The Target Type dialog is displayed.
16. Select **Service**.
The Target Services dialog is displayed.
17. Navigate to **XrefOrderApp, References, Logger**.
18. Select **Write** and click **OK**.
19. Click the **Transformation** icon next to the **Transform Using** field.
The Reply Transformation map dialog is displayed.
20. Select **Create New Mapper File** and enter EBS_TO_COMMON_UPDATE.xsl.
21. Click **OK**.
A EBS_TO_COMMON_UPDATE.xsl window is displayed.
22. Connect **db:X:APP_ID** source element to the **db:X:APP_ID**.
23. From the **File** menu, click **Save All** and close the EBS_TO_COMMON_UPDATE.xsl tab.
24. In the Synchronous Reply panel, click the **Assign Values** icon.
The Assign Values dialog is displayed.
25. Click **Add**.
The Assign Value dialog is displayed.
26. In the **From** section, select **Expression**.
27. Click the **Invoke Expression Builder** icon.
The Expression Builder dialog is displayed.
28. Enter following expression in the **Expression** field and click **OK**.

```
concat('UPDATE-', $in.OutputParameters/db:OutputParameters/db:X_APP_ID, '.xml')
```
29. In the **To** section, select **Property**.
30. Select **jca.file.FileName** property and click **OK**.
31. Click **OK**.
The update operation panel would appear as shown in [Figure 47-60](#).

Figure 47–60 Update Operation with EBS Target Service



32. From the **File** menu, click **Save All**.

47.7.1.10 Task 10: Configuring Oracle Application Server Connection

An Oracle Application Server connection is required for deploying your SOA composite application. For information on creating Oracle Application Server connection, refer to *Oracle Fusion Middleware User's Guide for Technology Adapters*.

47.7.1.11 Task 11: Deploying the Composite Application

Deploying the `XrefOrderApp` composite application to Oracle Application Server consists of following steps:

- Creating an Application Deployment Profile
- Deploying the Application to Oracle Application Server

For detailed information about these steps, see [Section 43.2, "Deploying a Single SOA Composite in Oracle JDeveloper"](#).

Using Two-Layer Business Process Management (BPM)

Two-Layer BPM enables you to create dynamic business processes whose execution, rather than being pre-determined at design time, depends on elements of the context in which the process executes. Such elements could include, for example, the type of customer, the geographical location, or the channel.

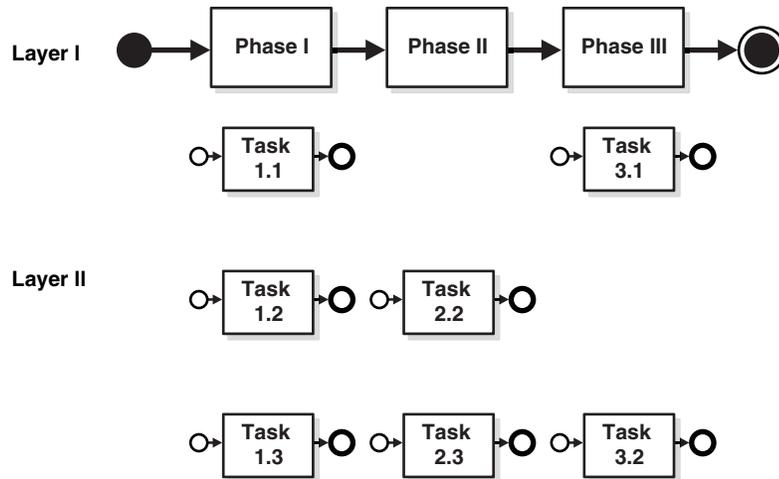
To illustrate further, suppose that you have an application that performs multi-channel banking using various processes. In this scenario, the execution of each process would depend on the channel for each particular process instance.

This chapter includes the following sections:

- [Section 48.1, "Introduction to Two-Layer Business Process Management"](#)
- [Section 48.2, "Phase Activities"](#)
- [Section 48.3, "The Dynamic Routing Decision Table"](#)
- [Section 48.4, "Use Case: Two-Layer BPM"](#)

48.1 Introduction to Two-Layer Business Process Management

Two-Layer BPM enables you to model business processes using a layered approach. In that model, a first level is a very abstract specification of the business process. Activities of a first-level process delegate the work to processes or services in a second level. [Figure 48-1](#) illustrates this behavior.

Figure 48–1 Two-Layer BPM

In [Figure 48–1](#), the Phase I activity of the business process can delegate its work to one of the corresponding Layer II processes: Task 1.1, Task 1.2 or Task 1.3.

The two-layer BPM functionality enables you to create the key element—namely, the phase activity—declaratively.

By using the DT@RT functionality of Oracle Business Rules, you can add more channels dynamically without having to re-deploy the business process. DT@RT enables you to add rules (columns) to the dynamic routing decision table at runtime. Then, during runtime, business process instances consider those new rules and eventually route the requests to a different channel.

The DT@RT functionality of Oracle Business Rules also enables you to modify the end-point reference of a service that is invoked from a phase activity, pointing that reference to a different service.

Note: In Oracle Fusion Middleware 11g Release 1 (11.1.1), you can use the DT@RT functionality of Oracle Business Rules only by way of the Oracle Business Rules SDK.

For information about using the Oracle Business Rules SDK, see:

- *Oracle Fusion Middleware User's Guide for Oracle Business Rules*
 - *Oracle Fusion Middleware Java API Reference for Oracle Business Rules*
-

To enable Two-Layer BPM, you follow these steps:

Table 48–1 Steps for Enabling Two-Layer BPM

Step	Information
Install the Oracle WebLogic Server	<i>Oracle WebLogic Server Installation Guide</i>
Design the SOA composite application	Section 4.2.1, "How to Create an Application and a Project"
Create Element-type variables named <code>phaseIn</code> and <code>phaseOut</code>	" Creating Variables " on page 48-9
Create a Phase Activity	Section 48.2, "Phase Activities" on page 48-3

Table 48–1 (Cont.) Steps for Enabling Two-Layer BPM

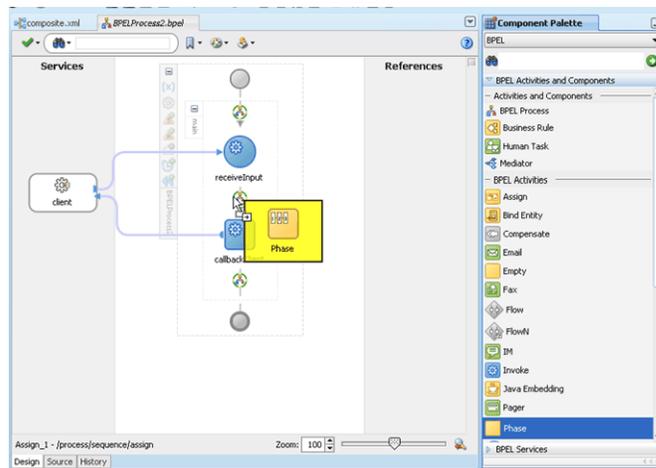
Step	Information
Create and Edit the Dynamic Routing Decision Table	Section 48.3, "The Dynamic Routing Decision Table" on page 48-5
Add Assign activities to the BPEL process model	Section A.2.2, "Assign Activity"
Create the Application Deployment Profile	Chapter 43, "Deploying SOA Composite Applications"
Create an Application Server Connection	Section 43.1, "Creating an Application Server Connection"
Deploy the Application	Chapter 43, "Deploying SOA Composite Applications"

48.2 Phase Activities

In Two-Layer BPM, a Phase is a level-1 activity in the BPEL process model. It complements the existing higher-level BPEL activities Business Rule and Human Task.

48.2.1 Creating a Phase Activity

You add a Phase to a process declaratively by using the BPEL Designer in Oracle JDeveloper just as you would any other BPEL activity—by dragging and dropping it from the BPEL Activities and Components palette to the process model.

Figure 48–2 Phase Activity in BPEL Designer

Note: The reference WSDL (Layer 2 or Called References) must have the same abstract WSDL as that for the Phase reference that gets auto-created.

48.2.2 How to Create a Phase Activity

You create the Phase activity for your composite application after you have created the necessary variables as described in ["Creating Variables"](#) on page 48-9.

Once the Phase is dropped into the level-1 BPEL process, a wizard guides you through the various configuration steps. It first displays the Create Phase Activity dialog, in which you specify the input and the output of the Phase.

To create a Phase activity:

In the Create Phase Activity dialog:

1. Enter a value in the **Name** field.
2. Select the **Inputs and Outputs** icon, which is a green plus sign (+). The Variable Chooser dialog appears.
3. Select **Process**, **Variables**, and **phaseIn**, and then click OK. The Phase dialog box is displayed with the phaseIn variable populated.
4. From the **Inputs and Outputs** icon, select **Select Output**. The Variable Chooser dialog box appears.
5. Select **Process**, **Variables**, and **phaseOut**.
6. Click **OK**. The Phase dialog box is displayed with the input and output variable names populated.
7. Click OK. The BPEL Designer displays the `.bpel` page for your process.
8. Click **Save All** from the File menu. Close the `.bpel` page.
9. Click the `composite.xml` page. The SCA composite diagram appears.

48.2.3 What Happens When You Create a Phase Activity

When you create a Phase activity, the artifacts described in [Table 48–2](#) are created.

Table 48–2 *Artifacts Created with a Phase Activity*

Artifact	Description
BPEL scope	At the location where the user dropped the phase activity in the BPEL process model a new BPEL scope is created and inserted into the BPEL process. The scope has the name of the phase activity. Within the scope, a bunch of standard BPEL activities are created. The most important ones are one invoke activity to a mediator and one receive activity from the mediator.
Mediator component	With the SCA composite of the BPEL component, a new Mediator component is created and wired to the phase activity of the BPEL component that comprises the level-1 BPEL process where the phase activity has been dropped into the process model. The input and output of the Mediator component is defined by the input and output of the phase activity. The mediator plan (this are the processing instructions of the mediator component) is very simple; it delegates creation of the processing instructions to the business rules component.

Table 48–2 (Cont.) Artifacts Created with a Phase Activity

Artifact	Description
Business Rules component	<p>Within the SCA composite of the BPEL component, a new Business Rules component is created and wired to the mediator component associated with the Phase activity of the BPEL process. The business rule component includes a rule dictionary. The rule dictionary contains metadata for such rule engine artifacts as fact types, rule sets, rules, decision tables, and similar artifacts. As part of creating the business rules component the rule dictionary is pre-initialized with the following data:</p> <ul style="list-style-type: none"> ■ Fact Type Model: The data model that can be used for modeling rules. The rule dictionary will be populated with a fact type model that corresponds to the input of the phase activity together with some fixed data model that is required as part of the contract between the mediator and the business rules component. ■ Ruleset: A container of rules that is used as a kind of grouping mechanism for rules. A ruleset can be exposed as a service. One ruleset is created within the rule dictionary. ■ Decision Table: From a rules engine perspective, a decision table is simply a collection of rules with the same fact type model elements in the condition and action part of the rules so that the rules can be visualized in a tabular format. The new decision table is created within the ruleset. ■ Decision Service: A decision service is created that exposes the ruleset as a service of the business rules SCA component. The service interface is used by the mediator to evaluate the decision table.

48.2.4 What Happens at Runtime When You Create a Phase Activity

At runtime, the input of the Phase activity is used to evaluate the dynamic routing decision table. This is performed by a specific decision component of the Phase activity. The result of this evaluation is an instruction for the Mediator. The Mediator routes the request to a service based on instructions from the decision component.

Note: Phase activity is asynchronous in nature. Synchronous or one-way Phase activity is not possible.

48.2.5 What You May Need to Know About Creating a Phase Activity

When creating a phase activity, you need to know the following:

- Rules that you will need to either configure or create in the decision service. This will be based on data from payload that you will use to evaluate a rule.
- For each rule created in the decision service, you need to know the corresponding endpoint URL that needs to be invoked when a rule evaluates to true. This endpoint URL will be used by Mediator to invoke the service in layer 2.

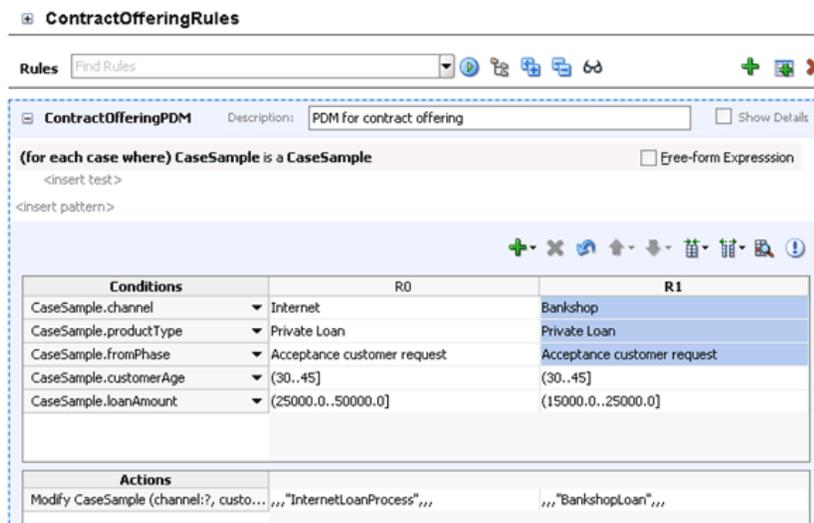
48.3 The Dynamic Routing Decision Table

A Dynamic Routing Decision Table is a decision table evaluated by Business Rules. Conditions are evaluated on the input data of a Phase activity. The result of the evaluation is routing instruction for the Mediator.

48.3.1 How to Create the Routing Decision Table

After you have created the Phase activity, the wizard launches the Rule Designer in Oracle JDeveloper for you edit the Routing Decision Table. Figure 48–3 shows a sample decision table within rule designer.

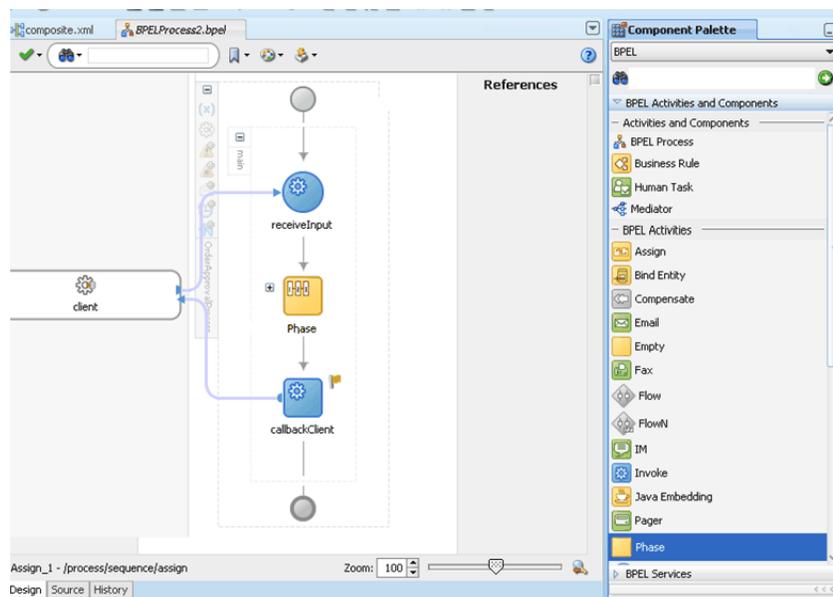
Figure 48–3 Sample Decision Table



You can leave the decision table empty while modeling the level-2 process phases, and complete it after the level-1 process is being deployed using the Business Analyst tool.

Once you have created and edited the Routing Decision Table, the new level-1 phase activity appears in the BPEL process in Oracle JDeveloper as illustrated in Figure 48–4.

Figure 48–4 Completed Level-1 Phase in Oracle JDeveloper



48.3.2 What Happens When You Create the Routing Decision Table

By creating the Routing Decision Table, you are configuring the decision service to dynamically evaluate the conditions applied to the incoming payload and give the corresponding routing rules to the Mediator. The Mediator then executes these rules when invoking the service in layer 2.

More specifically, here is what happens at design time when you create the Routing Decision Table:

- A new decision component is created in the composite of the project.
- A new rule dictionary is created in the composite project directory.
- The rule dictionary is populated with a data model that reflects the data model of the phase input—that is, the XML schema of the Phase input is imported into the rule dictionary.

Note: No Transform, Assign or Validation can be performed on a payload.

48.4 Use Case: Two-Layer BPM

This section contains a use case for a sales process. Steps to run the use case are as follows:

1. Install the server as described in *Oracle WebLogic Server Installation Guide*.
2. Model the sample by performing these tasks:
 - a. Designing the SOA composite as described in [Section 48.4.1, "Designing the SOA Composite"](#)
 - b. [Section 48.4.2, "Creating a Phase Activity"](#)
 - c. [Section 48.4.3, "Creating and Editing the Dynamic Routing Decision Table"](#)
 - d. Adding assign activities to the BPEL process model as described in [Section 48.4.4, "Adding Assign Activities to the BPEL Process Model"](#)
3. Deploy the sample with JDeveloper as described in [Section 48.4.5, "Deploying the Sample with JDeveloper"](#).

48.4.1 Designing the SOA Composite

You design the SOA composite application in JDeveloper.

To design the SOA composite:

1. In JDeveloper, select **New** from the **File** menu. The New Gallery dialog box appears.
2. Click **OK**. The Create Application dialog box appears.
3. Enter **LoanFlowRouterApp** in the Application Name field, and then click **OK**. The Create Project dialog box appears.
4. Click **Cancel** in the Create Project dialog box.
5. Right-click the LoanFlowRouterApp menu and select **New Project**. The New Gallery dialog box appears.

6. Select SOA Project from the Items list and click **OK**. The Create SOA Project dialog box appears.
7. Enter **LoanFlowRouter** in the Project Name field and select **Composite With BPEL** in the Composite Template list, and click **OK**. The Create BPEL Process page appears.
8. Enter **LoanFlowRouterProcess** in the Name field of the Create BPEL Process page and select Asynchronous BPEL Process from the Template list. Click **OK**.
9. Import the `AutoLoanTypes.xsd` schema into the project `xsd` folder. The `AutoLoanTypes.xsd` schema is as follows:

```
<?xml version="1.0"?>
<xsd:schema attributeFormDefault="qualified" elementFormDefault="qualified"
<xsd:schema attributeFormDefault="qualified" elementFormDefault="qualified"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.autoloan.com/ns/autoloan"
xmlns:tns="http://www.autoloan.com/ns/autoloan">
<xsd:element name="loanApplication" type="tns:LoanApplicationType"/>
<xsd:element name="loanOffer" type="tns:LoanOfferType"/>
<xsd:element name="invalidApplication" type="tns:InvalidApplicationType"/>
<xsd:element name="loan" type="tns:LoanType"/>
<xsd:complexType name="InvalidApplicationType">
<xsd:sequence>
<xsd:element name="error" type="xsd:string"/>
<xsd:element name="application" type="tns:LoanApplicationType"/>
</xsd:sequence>
</xsd:complexType>
<xsd:complexType name="LoanType">
<xsd:sequence>
<xsd:element ref="tns:loanApplication"/>
<xsd:element ref="tns:loanOffer"/>
</xsd:sequence>
</xsd:complexType>
<xsd:complexType name="LoanApplicationType">
<xsd:sequence>
<xsd:element name="SSN" type="xsd:string"/>
<xsd:element name="email" type="xsd:string"/>
<xsd:element name="customerName" type="xsd:string"/>
<xsd:element name="loanAmount" type="xsd:double"/>
<xsd:element name="carModel" type="xsd:string"/>
<xsd:element name="carYear" type="xsd:string"/>
<xsd:element name="creditRating" type="xsd:int"/>
</xsd:sequence>
</xsd:complexType>
<xsd:complexType name="LoanOfferType">
<xsd:sequence>
<xsd:element name="providerName" type="xsd:string"/>
<xsd:element name="selected" type="xsd:boolean"/>
<xsd:element name="approved" type="xsd:boolean"/>
<xsd:element name="APR" type="xsd:double"/>
</xsd:sequence>
</xsd:complexType>
</xsd:schema>
```

10. After importing the `AutoLoanTypes.xsd` schema, open the `LoanFlowRouterProcess.bpel` page.

Creating Variables

Note: Phase variables can be only of the Element type.

1. Click the **Variables...** icon. The Variables dialog box appears.
2. Click the **Create...Alt+N** icon. The Create Variable dialog box appears.
3. Enter **phaseIn** in the Name field. Click the **Element** option.
4. Click the **Browse Elements** icon. The Type Chooser dialog box is displayed.
5. Select **Project Schema Files, AutoLoanTypes.xsd, and loanApplication**, and then click **OK**. The Create Variable dialog box appears with the element name populated.
6. Click **OK**. The Variables dialog box is displayed with the variable name populated.
7. Click the **Create...Alt+N** icon in the Variables dialog box. The Create Variable dialog appears.
8. Enter **phaseOut** in the **Name** field. Click the **Element** option. Click the **Browse Elements** icon. The Type Chooser dialog appears.
9. Select **Project Schema Files, AutoLoanTypes.xsd, and loanOffer**, and then click **OK**. The Create Variable dialog box appears with the element name populated.
10. Click **OK**. The Variables Dialog box is displayed with the input and output variable names populated.
11. Click **OK**. The variables have been created and the `LoanFlowRouterProcess.bpel` page appears.

48.4.2 Creating a Phase Activity

You create the Phase activity by using JDeveloper.

To create a Phase activity:

1. Click the `LoanFlowRouterProcess.bpel` page. Drag and drop a Phase activity from the BPEL component palette into the process model. The Phase dialog box appears.
2. Enter **routeToLoanApplication** in the **Name** field. From the **Inputs and Outputs** plus icon, select **Select Input**. The Variable Chooser dialog box appears.
3. Select **Process, Variables, and phaseIn**, and then click **OK**. The Phase dialog box is displayed with the phaseIn variable populated.
4. Select **Select Output** from the **Inputs and Outputs** plus icon. The Variable Chooser dialog box is displayed.
5. Select **Process, Variables, and phaseOut**.
6. Click **OK**. The Phase dialog box is displayed with the input and output variable names populated.
7. Click **OK**. The `LoanFlowRouterProcess.bpel` page is displayed.
8. Click **Save All** from the **File** menu. Close the `LoanFlowRouterProcess.bpel` page.
9. Click the `composite.xml` page. The SCA composite diagram is displayed.

Note:

- As part of the Phase activity wizard, three components are created: Rules, Mediator, and Dynamic Reference.
 - The Rules component returns an executable case for the Mediator component, on the basis of the rules defined.
 - The Mediator component routs on the basis of the routing rules received from the Rules component.
 - The Dynamic Reference component is the dummy reference for the second-level processes.
 - The Rule dictionary is populated with the fact type model of the Mediator and the fact type corresponding to the input of the Phase activity, which in this case is `LoanApplicationType`.
 - An empty decision table called the `RoutingTable` is created that needs to be edited providing dynamic routing rules.
-

48.4.3 Creating and Editing the Dynamic Routing Decision Table

You create and edit the Dynamic Routing Decision Table by using Oracle JDeveloper.

To create and edit the Dynamic Routing Decision Table:

1. Open the `LoanFlowRouterProcess.bpel` page. Double-click the **Phase** activity in the process diagram. The Phase dialog box appears.
2. Click the **Edit Dynamic Rules** button. The Rule Designer page appears.
3. Click **Ruleset_1** under **Rulesets** from the **Types** list. The `Ruleset_1` page with an empty `RoutingTable` appear.
4. Click the arrow under **Conditions** and select **loanAmount** from the list. The text field above the list is populated with `LoanApplicationType.loanAmount`.
5. Right-click the `LoanApplicationType.loanAmount` condition and select **Edit Local List of Ranges**. The **Edit Bucketset** dialog box is displayed.
6. Click the **Click to add "Less Than Or Equal" Range-Position** icon (first blue icon) beside Range Editor. The value **0** is displayed at the center of the range axis (the default range is -10000 to 100000).
7. Enter **0** in the **Minimum** field and **100000** in the **Maximum** field. The **Minimum** and **Maximum** fields are populated with the new values.
8. Click one of the interval icons in the Range Editor and create a range of **0** to **200000**.

Tip: To adjust a range, move it with the mouse along the range axis or click into the range icon, type a value, and press **Enter** on the keyboard.

9. Click one of the interval icons in the Range Editor and create a range of **200000** to **500000**. Click **OK**. The `RoutingTable` page is displayed.
10. Right-click the `LoanApplicationType.loanAmount` condition and select **Split Condition**. The new rule columns, such as **R1**, **R2**, **R3** and **R4**, are displayed with conditions according to the bucketset definition of the `loanAmount` attribute.

11. Click the plus icon in the RoutingTable page and select **Add Actions** and then assert new from the list. The Create Action dialog box is displayed.
12. Enter some default values for the fact type attributes. Select the **Parameterized** option for the serviceBindingInfo attribute.
13. Click **OK**. The RoutingTable page is displayed with Actions defined for the rules R1-R4.
14. Enter the values for the parametrized attributes of the Actions. See [Table 48–3](#) for the values to be entered. This completes editing the RoutingTable page.

Table 48–3 Attributes for the RoutingAction Fact Type

Attribute	Default	Fixed	Parametrized	Required	Description
caseName		No	Yes	No	Some descriptive text (used in Mediator mplan)
cbkOperation	null	Yes	No	No	
executionType	direct	Yes	No	Yes	Execution type can be “direct” or “queued”
onCbkOperation		No		Yes	Callback operation
serviceBindingInfo		No	Yes	Yes	Service endpoint URL
serviceOperation		No	Yes	Yes	Service operation
serviceReference		No	Yes	Yes	Service reference

15. Click the `LoanFlowRouterProcess.bpel` page. Click **OK** in the Phase dialog.
16. Click the `composite.xml` page. The SCA composite diagram appears after the Routing Table has been created and edited.
17. Click **Save All** from the **File** menu. Close the `composite.xml` page.

48.4.4 Adding Assign Activities to the BPEL Process Model

Before deploying the Phase activity, you must initialize the Phase variables. You do this by adding Assign activities in the phase in the BPEL process.

To add Assign activities to the BPEL Process Model

1. Click the `LoanFlowRouterProcess.bpel` page. Drag and drop an **Assign** activity from the BPEL component palette into the process model between the **receiveInput** activity and the **Phase** activity. The **Assign** activity is added to the process model.
2. Double-click the **Assign** activity. The Assign dialog box is displayed.
3. Enter **AssignInput** in the **Name** field in the General tab.
4. Select the Copy Operation tab. Click the plus icon and select Copy Operation from the list. The Create Copy Operation dialog appears.
5. Create an input copy operation for **12345**.
6. Similar to Step 5, create the copy operations mentioned in the following table:

Table 48–4 Copy Operations for Adding Assign Activities

From	To
'scott.tiger@oracle.com'	phaseIn/ns1:loanApplication/ns1:email
inputVariable/payload/client:LoanFlowRouterProcessProcessRequest/client:input	phaseIn/ns1:ratingrequest/ns1:customerName
number(15000.0)	phaseIn/ns1:loanApplication/ns1:loanAmount
'BMW'	phaseIn/ns1:loanApplication/ns1:carModel
'2000'	phaseIn/ns1:loanApplication/ns1:carYear
number(300)	phaseIn/ns1:loanApplication/ns1:creditRating

7. Click **OK** in the Create Copy Operation dialog. The Assign dialog box appears with the input copy operation values populated.
8. Click **OK**. The `LoanFlowRouterProcess.bpel` page is displayed again.
9. Drag and drop another **Assign** activity from the BPEL component palette into the process model between the **Phase** activity and the **callbackClient** activity. The new **Assign** activity is added to the process model.
10. Double-click the **Assign** activity. The Assign dialog box is displayed.
11. Enter **AssignOutput** in the Name field in the General tab.
12. Select the Copy Operation tab. Click the plus icon and select Copy Operation from the list. The Create Copy Operation dialog appears.
13. Create the output copy operation.
14. Click **OK** in the Create Copy Operation dialog box. The Assign dialog box is displayed with the output copy operation value populated.
15. Click **OK**. The `LoanFlowRouterProcess.bpel` page appears after the input and output Assign activities are created.
16. Click **Save All** from the File menu.

48.4.5 Deploying the Sample with JDeveloper

You need to deploy the application profile for the SOA project and application you created in the earlier steps. Steps to deploy the profile using JDeveloper are:

- [Creating an Application Deployment Profile](#)
- [Creating an Application Server Connection](#)
- [Deploying the Application](#)

48.4.5.1 Creating an Application Deployment Profile

To create an application deployment profile:

1. Click the Application Menu dropdown adjacent to the `LoanFlowRouterApp` project and select **Application Properties**. The Application Properties dialog box is displayed.
2. Select **Deployment**. The Application Properties dialog box with the Deployment page appears on the right pane of the dialog.

3. Click **New**. The Create Deployment Profile dialog box is displayed.
4. Select **OAR File** from the Archive Type, and enter **phaseActivity** in the **Name** field.
5. Click **OK**. The name of the deployment profile you created appears in the Deployment Profiles pane.
6. Double-click **phaseActivity** in the Deployment Profiles pane. The OAR Deployment Profile Properties dialog box is displayed.
7. Click **Application Assembly**, and select **sca_LoanFlowRouter**, and click **OK**.
8. Click **OK**. You have created the deployment profile with the name phaseActivity.

48.4.5.2 Creating an Application Server Connection

You need to establish connectivity between the design-time environment and the server on which you want to deploy it.

To create an application server connection:

1. From the **File** main menu, select **New > Connections > Application Server Connection**.
2. Click **OK**.
3. In the **Connection Name** field, enter a connection name.
4. From the **Connection Type** list, select **WebLogic 10.3**.
5. Click **Next**.
6. In the **Username** field, enter `weblogic`.
7. In the **Password** field, enter the password for connecting to the application server.
8. Click **Next**.
9. Enter the hostname for the application server that is configured with the SOA Infrastructure.
10. In the **WLS Domain** field, enter the Oracle WebLogic Server domain.
11. Click **Next**.
12. Click **Test Connection**. If the test is successful, a message informs you of this.
13. Click **Finish**.
14. From the **File** menu, select **Save All**.

48.4.5.3 Deploying the Application

You are now ready to deploy the composite application to Oracle WebLogic Server.

To deploy the application:

1. Click the **Application Menu** dropdown and select **Deploy**, *deployment_profile_name*, **to**, *appserver_connection_name*.
2. Click **OK** in the Revision ID dialog box.
3. Click **OK** in the Deployment Plan dialog box.

Testing SOA Composite Applications

This chapter describes how to create, deploy, and run test cases that automate the testing of SOA composite applications. Test cases enable you to simulate the interaction between a SOA composite application and its web service partners before deployment in a production environment. This helps to ensure that a process interacts with web service partners as expected by the time it is ready for deployment to a production environment.

This chapter includes the following sections:

- [Section 49.1, "Introduction to the Composite Test Framework"](#)
- [Section 49.2, "Introduction to the Components of a Test Suite"](#)
- [Section 49.3, "Creating Test Suites and Test Cases"](#)
- [Section 49.4, "Creating the Contents of Test Cases"](#)
- [Section 49.5, "Deploying and Running a Test Suite"](#)

49.1 Introduction to the Composite Test Framework

Oracle SOA Suite provides an automated test suite framework for creating and running repeatable tests on a SOA composite application.

The test suite framework provides the following features:

- Simulates web service partner interactions
- Validates process actions with test data
- Creates reports of test results

49.1.1 Test Cases Overview

The test framework supports testing at the SOA composite application level. In this type of testing, wires, service binding components, service components (such as BPEL processes and Oracle Mediator service components), and reference binding components are tested.

For more information, see [Section 49.3, "Creating Test Suites and Test Cases."](#)

49.1.2 Test Suites Overview

Test suites consist of a logical collection of one or more test cases. Each test case contains a set of commands to perform as the test instance is executed. The execution of a test suite is known as a test run. Each test corresponds to a single SOA composite application instance.

For more information, see the following:

- [Section 49.3, "Creating Test Suites and Test Cases"](#)
- [Section 49.4, "Creating the Contents of Test Cases"](#)

49.1.3 Emulations Overview

Emulations enable you to simulate the behavior of the following components with which your SOA composite application interacts during execution:

- Internal service components inside the composite
- Binding components outside the composite

Instead of invoking another service component or binding component, you can specify a response from the component or reference.

For more information, see the following:

- [Section 49.2.2, "Emulations"](#)
- [Section 49.4, "Creating the Contents of Test Cases"](#)

49.1.4 Assertions Overview

Assertions enable you to verify variable data or process flow. You can perform the following types of assertions:

- Entire XML document assertions:
Compare the element values of an entire XML document to the expected element values. For example, compare the exact contents of an entire loan request XML document to another document. The `XMLTestCase` class in the `XMLUnit` package includes a collection of methods for performing assertions between XML files. For more information about these methods, visit the following URL:
<http://xmlunit.sourceforge.net>
- Part section of message assertions:
Compare the values of a part section of a message to the expected values. An example is a payload part of an entire XML document message.
- Nonleaf element assertions:
Compare the values of an XML fragment to the expected values. An example is a loan application, which includes leaf elements `SSN`, `email`, `customerName`, and `loanAmount`.
- Leaf element assertions:
Compare the value of a selected string or number element or a regular expression pattern to an expected value. An example is the `SSN` of a loan application.

For more information about asserts, see [Section 49.2.3, "Assertions."](#)

49.2 Introduction to the Components of a Test Suite

This section describes and provides examples of the test components that comprise a test case. Methods for creating and importing these tests into your process are described in subsequent sections of this chapter.

49.2.1 Process Initiation

You first define the operation of your process in a binding component service such as a SOAP web service. [Example 49-1](#) defines the operation of `initiate` to initiate the TestFwk SOA composite application. The initiation payload is also defined in this section:

Example 49-1 Process Initiation

```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- Generated by Oracle SCA Test Modeler version 1.0 at [6/13/07 10:50 AM]. -->
<compositeTest compositeDN="TestFwk"
  xmlns="http://xmlns.oracle.com/sca/2006/test">
  <about></about>
  <initiate serviceName="client" operation="initiate" isAsync="true">
    <message>
      <part partName="payload">
        <content>
          <loanApplication xmlns="http://www.autoloan.com/ns/autoloan">
            <SSN>111222333</SSN>
            <email>joe.smith@oracle.com</email>
            <customerName>Joe Smith</customerName>
            <loanAmount>20000</loanAmount>
            <carModel>Camry</carModel>
            <carYear>2007</carYear>
            <creditRating>800</creditRating>
          </loanApplication>
        </content>
      </part>
    </message>
  </initiate>
</compositeTest>
```

49.2.2 Emulations

You create emulations to simulate the message data that your SOA composite application receives from web service partners.

In the test code in [Example 49-2](#), the loan request is initiated with an error. A fault message is received in return from a web service partner:

Example 49-2 Emulations

```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- Generated by Oracle SCA Test Modeler version 1.0 at [7/3/07 3:29 PM]. -->
<compositeTest compositeDN="CompositeTest"
  xmlns="http://xmlns.oracle.com/sca/2006/test">
  <about></about>
  <initiate serviceName="client" operation="initiate" isAsync="true">
    <message>
      <part partName="payload">
        <filePath>loanApplication.xml</filePath>
      </part>
    </message>
  </initiate>
  <wireActions wireSource="LoanBroker/CreditRatingService" operation="process">
    <emulate duration="PT0S">
      <fault faultName="ser:NegativeCredit" xmlns:ser="http://services.otn.com">
        <message>
          <part partName="payload">
```

```

        <filePath>creditRatingFault.xml</filePath>
    </part>
</message>
</fault>
</emulate>
</wireActions>
</compositeTest>

```

Two message files, `loanApplication.xml` and `creditRatingFault.xml`, are invoked in this emulation. If the loan application request in `loanApplication.xml` contains a social security number beginning with 0, the `creditRatingFault.xml` file returns the fault message shown in [Example 49-3](#):

Example 49-3 Fault Message

```

<error xmlns="http://services.otn.com">
  Invalid SSN, SSN cannot start with digit '0'.
</error>

```

For more information, see [Section 49.4, "Creating the Contents of Test Cases."](#)

49.2.3 Assertions

You create assertions to validate an entire XML document, a part section of a message, a nonleaf element, or a leaf element at a point during SOA composite application execution. [Example 49-4](#) instructs Oracle SOA Suite to ensure that the content of the `customerName` variable matches the content specified.

Example 49-4 Assertions

```

<?xml version="1.0" encoding="UTF-8" ?>
<!-- Generated by Oracle SCA Test Modeler version 1.0 at [6/13/07 10:51 AM]. -->
<compositeTest compositeDN="TestFwk"
  xmlns="http://xmlns.oracle.com/sca/2006/test">
  <about></about>
  <initiate serviceName="client" operation="initiate" isAsync="true">
    <message>
      <part partName="payload">
        <content>
          <loanApplication xmlns="http://www.autoloan.com/ns/autoloan">
            <SSN>111222333</SSN>
            <email>joe.smith@oracle.com</email>
            <customerName>Joe Smith</customerName>
            <loanAmount>20000</loanAmount>
            <carModel>Camry</carModel>
            <carYear>2007</carYear>
            <creditRating>800</creditRating>
          </loanApplication>
        </content>
      </part>
    </message>
  </initiate>
  <wireActions wireSource="client" operation="initiate">
    <assert comparisonMethod="string">
      <expected>
        <location key="input" partName="payload"
          xpath="/s1:loanApplication/s1:customerName"
          xmlns:s1="http://www.autoloan.com/ns/autoloan"/>
          <simple>Joe Smith</simple>
        </expected>

```

```
</assert>
</wireActions>
</compositeTest>
```

For more information, see [Section 49.4, "Creating the Contents of Test Cases."](#)

49.2.4 Message Files

Message instance files provide a method for simulating the message data received back from web service partners. You can manually enter the received message data into this XML file or load a file through the test mode of the SOA Composite Editor. For example, the following message file simulates a credit rating result of 900 returned from a partner:

```
<rating xmlns="http://services.otn.com">900</rating>
```

For more information about loading message files into test mode, see [Section 49.4, "Creating the Contents of Test Cases."](#)

49.3 Creating Test Suites and Test Cases

This section describes how to create test suites and their test cases for a SOA composite application. The test cases consist of sets of commands to perform as the test instance is executed.

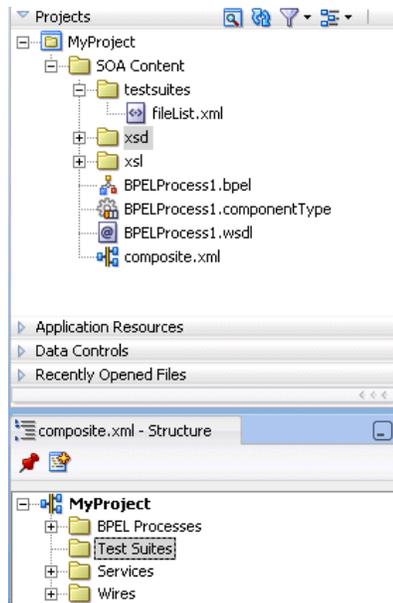
Note: Do *not* enter a multibyte character string as a test suite name or test case name. Doing so causes an error to occur when the test is executed from Oracle Enterprise Manager Fusion Middleware Control Console.

49.3.1 How to Create Test Suites and Test Cases

To create test suites and test cases:

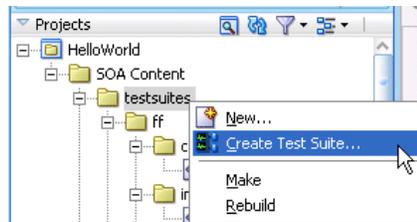
1. Open the SOA Composite Editor.
2. Open the SOA composite application in which to create a test suite.
3. Go to the Application Navigator or Structure window. If the Structure window shown in [Figure 49-1](#) does not appear, select **Structure** from the **View** main menu.

Figure 49–1 Structure Window



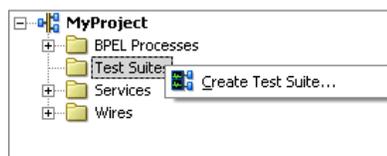
4. Create a test suite in either of two ways:
 - a. In the Application Navigator, right-click **testsuites** and select **Create Test Suite**. [Figure 49–2](#) provides details.

Figure 49–2 Create Test Suite Selection



- b. In the Structure window, right-click **Test Suites** and select **Create Test Suite**. [Figure 49–3](#) provides details.

Figure 49–3 Create Test Suite Selection



5. Enter a test suite name (for example, `logicTest`).
6. Click **OK**.
The Create Composite Test dialog appears.
7. Enter a test name (for this example, `TestDelivery` is entered) and an optional description. This description displays in the **Description** column of the Test Cases page of the **Unit Tests** tab in Oracle Enterprise Manager Fusion Middleware Control Console.

8. Click OK.

This action creates a test named **TestDelivery.xml** in the **Applications Navigator**, along with the following subfolders:

- **componenttests**

This folder is not used in 11g Release 1.

- **includes**

This folder is not used in 11g Release 1.

- **messages**

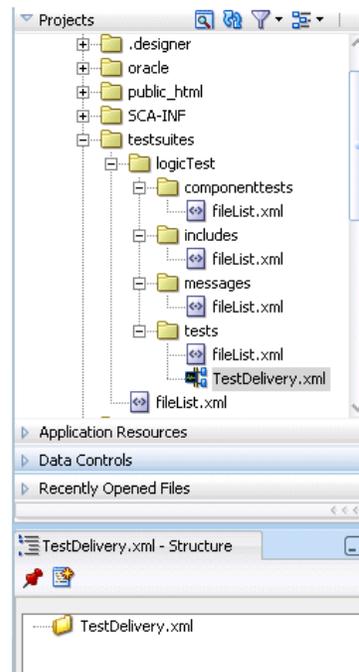
Contains message test files that you load into this directory through the test mode user interface.

- **tests**

Contains **TestDelivery.xml**.

A **TestDelivery.xml** folder also displays in the Structure window. [Figure 49–4](#) provides details. This indicates that you are in the test mode of the SOA Composite Editor. You can create test initiations, assertions, and emulations in test mode. No other modifications, such as editing the property dialogs of service components or dropping service components into the editor, can be performed in test mode.

Figure 49–4 TestDelivery.xml Folder



The following operating system test suite directory is also created:

```
C:\JDeveloper\mywork\application_name\project_name\testsuites\logicTest
```

The following subdirectories for adding test files are created beneath `logicTest`: `componenttests`, `includes`, `messages`, and `tests`.

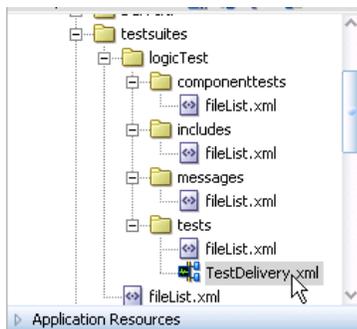
9. If you want to exit test mode and return to design mode in the SOA Composite Editor, click the last icon below **TestDelivery.xml** above the designer. [Figure 49-5](#) provides details.

Figure 49-5 Test Mode Exit



10. Save your changes when prompted.
11. Under the **testsuites** folder in the Application Navigator, double-click **TestDelivery.xml** to return to test mode. [Figure 49-6](#) provides details.

Figure 49-6 Test Mode Access



Notes:

- Do not edit the **filelist.xml** files that display under the subfolders of the **testsuites** folder. These files are automatically created during design time, and are used during runtime to calculate the number of test cases.
- You cannot create test suites within other test suites. However, you can organize a test suite into subdirectories.

49.4 Creating the Contents of Test Cases

Test cases consist of process initiations, emulations, and assertions. You add these actions to test cases in the test mode of the SOA Composite Editor. You create process initiations to initiate client inbound messages into your SOA composite application. You create emulations to simulate input or output message data, fault data, callback data, or all of these types that your SOA composite application receives from web service partners. You create assertions to validate entire XML documents, part sections of messages, nonleaf elements, and leaf elements as a process is executed.

49.4.1 How to Initiate Inbound Messages

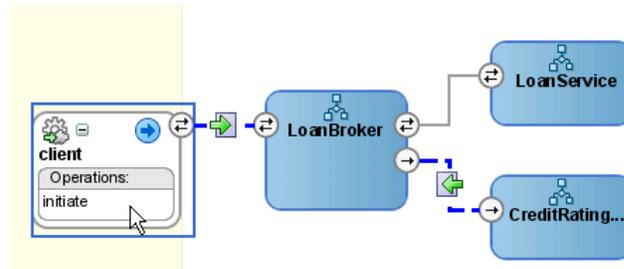
To initiate inbound messages:

You must first initiate the sending of inbound client messages to the SOA composite application.

1. Go to the SOA Composite application in test mode.

- Double-click the service binding component shown in [Figure 49-7](#) (for this example, named **initiate**).

Figure 49-7 Binding Component Service Access



The Edit Initiate dialog appears.

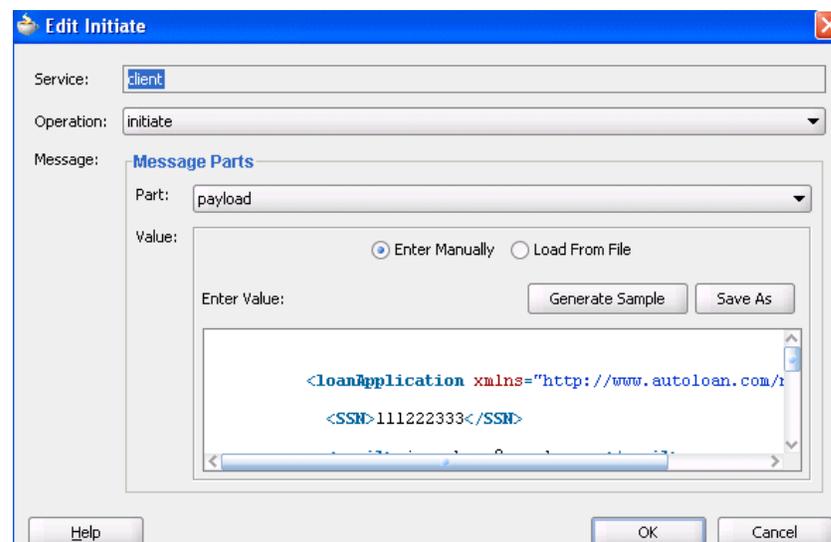
- Enter the details shown in [Table 49-1](#):

Table 49-1 Edit Initiate Dialog Fields and Values

Field	Value
Service	Displays the name of the binding component service (client).
Operation	Displays the operation type of the service binding component (initiate).
Part	Select the type of inbound message to send (for example, payload).
Value	Create a simulated message to send from a client: <ul style="list-style-type: none"> ■ Enter Manually Click to manually enter message data in the Enter Value field. A Generate Sample button enables you to automatically generate a sample file for testing. Click Save As to save the sample file. ■ Load From File Click the Browse icon to load message data from a file. The file is added to the messages folder in the Application Navigator.

[Figure 49-8](#) shows this dialog:

Figure 49-8 Edit Initiate Dialog



[Example 49–5](#) shows an inbound process initiation message from a client:

Example 49–5 Inbound Process Initiation Message

```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- Generated by Oracle SCA Test Modeler version 1.0 at [7/12/07 8:36 AM]. -->
<compositeTest compositeDN="CompositeTest"
xmlns="http://xmlns.oracle.com/sca/2006/test">
  <about/>
  <initiate serviceName="client" operation="initiate" isAsync="true">
    <message>
      <part partName="payload">
        <filePath>loanApplication.xml</filePath>
      </part>
    </message>
  </initiate>
  . . .
  . . .
```

The `loanApplication.xml` referenced in the process initiation file contains a loan application payload. [Example 49–6](#) provides details.

Example 49–6 Loan Application Payload

```
<loanApplication xmlns="http://www.autoloan.com/ns/autoloan">
  <SSN>111222333</SSN>
  <email>joe.smith@oracle.com</email>
  <customerName>Joe Smith</customerName>
  <loanAmount>20000</loanAmount>
  <carModel>Camry</carModel>
  <carYear>2007</carYear>
  <creditRating>800</creditRating>
</loanApplication>
```

4. Click **OK**.

49.4.2 How to Emulate Outbound Messages

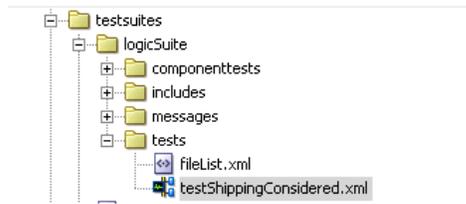
To emulate outbound messages:

Note: The creation of multiple emulations in an instance in a test case is supported only if one emulation is for an output message and the other is for a callback message.

You can simulate a message returned from a synchronous web service partner.

1. Go to the SOA composite application in test mode.
2. Beneath the **testsuites** folder in the Application Navigator, double-click a test case. [Figure 49–9](#) provides details.

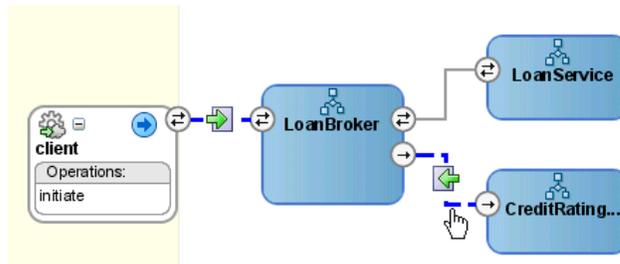
Figure 49–9 Test Case Access



The SOA composite application in the SOA Composite Editor is refreshed to display in test mode. This mode enables you to define test information.

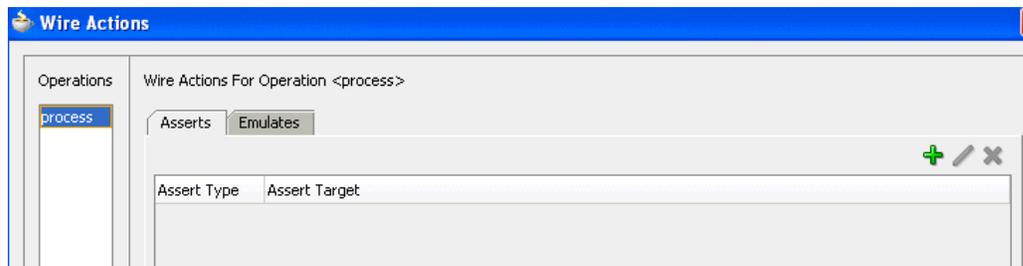
3. Double-click the wire of the SOA composite application area to test. For the example shown in [Figure 49–10](#), the wire between the **LoanBroker** process and the synchronous **CreditRating** web service is selected.

Figure 49–10 Wire Access



This displays the Wire Actions dialog shown in [Figure 49–11](#), from which you can design emulations and assertions for the selected part of the SOA composite application.

Figure 49–11 Wire Actions Dialog



4. Click the **Emulates** tab.
5. Click the **Add** icon.
6. Click **Emulate Output**.
7. Enter the details described in [Table 49–2](#):

Table 49–2 Emulate Output Message Dialog Fields and Values

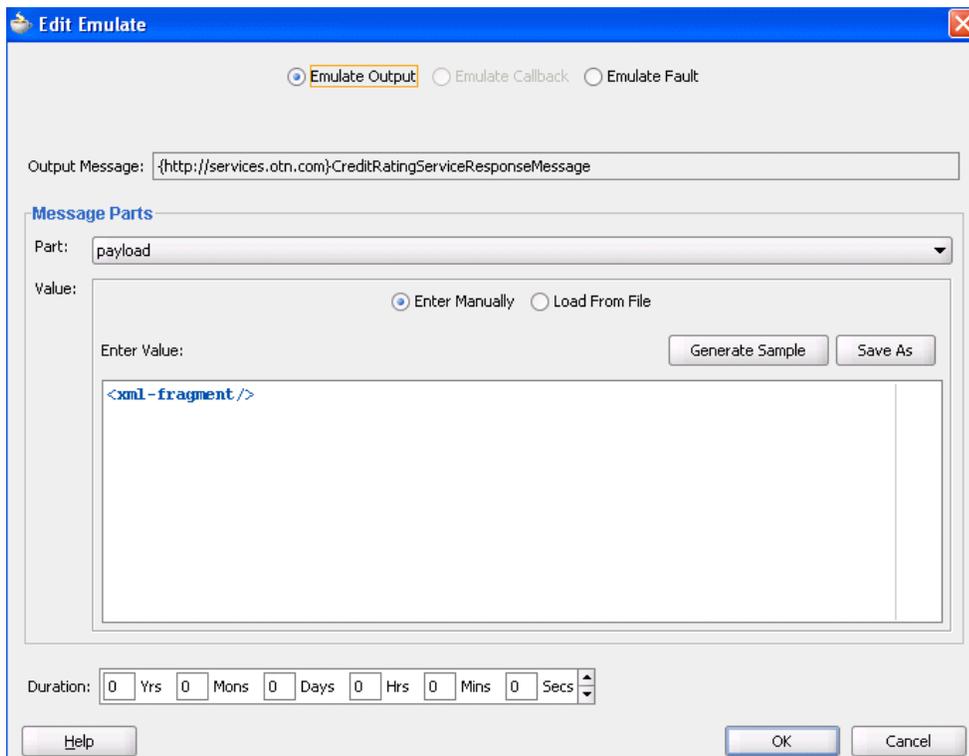
Field	Value
Part	Select the message part containing the output (for example, payload).
Value	Create a simulated output message to return from a web service partner:

Table 49–2 (Cont.) Emulate Output Message Dialog Fields and Values

Field	Value
<ul style="list-style-type: none"> Enter Manually 	Click to manually enter message data in the Enter Value field. A Generate Sample button enables you to automatically generate a sample file for testing. Click Save As to save the sample file.
<ul style="list-style-type: none"> Load From File 	Click the Browse icon to load message data from a file. The file is added to the messages folder in the Application Navigator.
Duration	Enter the maximum amount of time to wait for the message to be delivered from the web service partner.

Figure 49–12 shows this dialog:

Figure 49–12 Emulate Dialog with Emulate Output Selected



Example 49–7 shows a simulated output message from a synchronous web service partner that you enter manually or load from a file:

Example 49–7 Simulated Output Message Example

```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- Generated by Oracle SCA Test Modeler version 1.0 at [7/3/07 3:26 PM]. -->
<compositeTest compositeDN="CompositeTest"
xmlns="http://xmlns.oracle.com/sca/2006/test">
  <about></about>
  <initiate serviceName="client" operation="initiate" isAsync="true">
    <message>
      <part partName="payload">
        <filePath>loanApplication.xml</filePath>
      </part>
    </message>
```

```

</initiate>
<wireActions wireSource="LoanBroker/CreditRatingService" operation="process">
  <emulate duration="PT0S">
    <message>
      <part partName="payload">
        <filePath>creditRatingResult.xml</filePath>
      </part>
    </message>
  </emulate>
</wireActions>
</compositeTest>

```

The `creditRatingResult.xml` message file referenced in the output message provides details about the credit rating result.

```
<rating xmlns="http://services.otn.com">900</rating>
```

8. Click OK.

49.4.3 How to Emulate Callback Messages

To emulate callback messages:

Note: The creation of multiple emulations in an instance in a test case is supported only if one emulation is for an output message and the other is for a callback message.

You can simulate a callback message returned from an asynchronous web service partner.

1. Access the Wire Actions dialog by following Step 1 through Step 3 on page 49-10.
2. Click the **Emulates** tab.
3. Click the **Add** icon.
4. Click **Emulate Callback**. This field is only enabled for asynchronous processes.
5. Enter the details described in [Table 49-3](#):

Table 49-3 Emulate Callback Message Fields

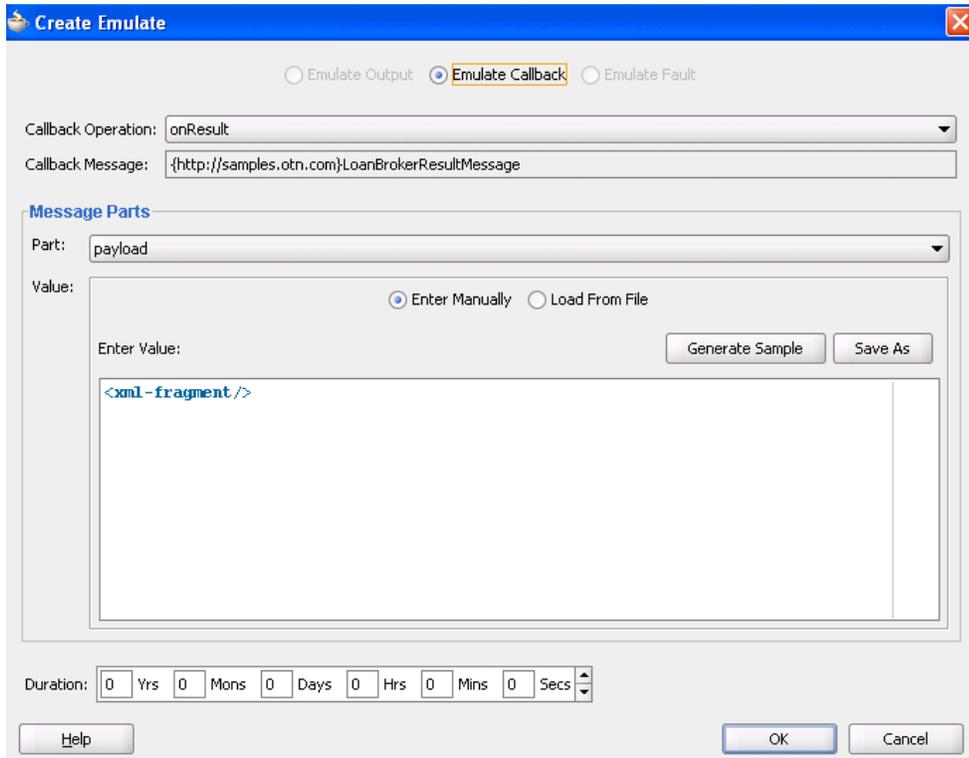
Field	Value
Callback Operation	Select the callback operation (for example, onResult).
Callback Message	Displays the callback message name of the asynchronous process.
Part	Select the message part containing the callback (for example, payload).
Value	Create a simulated callback message to return from an asynchronous web service partner: <ul style="list-style-type: none"> ■ Enter Manually Click to manually enter message data in the Enter Value field. A Generate Sample button enables you to automatically generate a sample file for testing. Click Save As to save the sample file. ■ Load From File Click the Browse icon to load message data from a file. The file is added to the messages folder in the Application Navigator.

Table 49-3 (Cont.) Emulate Callback Message Fields

Field	Value
Duration	Enter the maximum amount of time to wait for the callback message to be delivered from the web service partner.

Figure 49-13 shows this dialog:

Figure 49-13 Emulate Dialog with Emulate Callback Selected



Example 49-8 shows a simulated callback message from a web service partner. You enter this message manually or load it from a file:

Example 49-8 Simulated Callback Message Example

```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- Generated by Oracle SCA Test Modeler version 1.0 at [7/3/07 3:27 PM]. -->
<compositeTest compositeDN="CompositeTest"
xmlns="http://xmlns.oracle.com/sca/2006/test">
  <about></about>
  <initiate serviceName="client" operation="initiate" isAsync="true">
    <message>
      <part partName="payload">
        <filePath>loanApplication.xml</filePath>
      </part>
    </message>
  </initiate>
  <wireActions wireSource="LoanBroker/LoanService" operation="initiate">
    <emulate callbackOperation="onResult" duration="PT0S">
      <message>
        <part partName="payload">
```

```

        <filePath>loanOffer.xml</filePath>
    </part>
</message>
</emulate>
</wireActions>
</compositeTest>

```

The `loanOffer.xml` message file referenced in the callback message provides details about the credit rating approval. [Example 49–9](#) provides details.

Example 49–9 Credit Rating Approval Details

```

<loanOffer xmlns="http://www.autoloan.com/ns/autoloan">
  <providerName>Bank Of America</providerName>
  <selected>false</selected>
  <approved>true</approved>
  <APR>1.9</APR>
</loanOffer>

```

6. Click OK.

49.4.4 How to Emulate Fault Messages

To emulate fault messages:

You can simulate a fault message returned from a web service partner. This simulation enables you to test fault handling capabilities in your process.

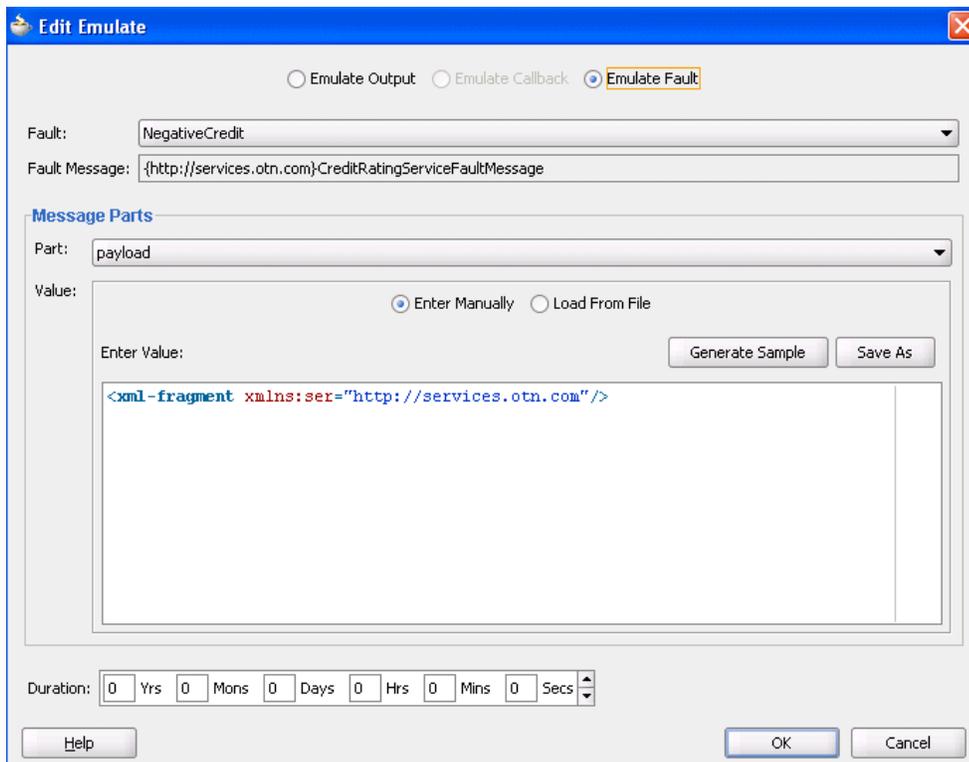
1. Access the Wire Actions dialog by following Step 1 through Step 3 on page 49-10.
2. Click the **Emulates** tab.
3. Click the **Add** icon.
4. Click **Emulate Fault**.
5. Enter the details described in [Table 49–4](#):

Table 49–4 Emulate Fault Message Fields

Field	Value
Fault	Select the fault type to return from a partner (for example, NegativeCredit).
Fault Message	Displays the message name.
Part	Select the message part containing the fault (for example, payload).
Value	Create a simulated fault message to return from a web service partner: <ul style="list-style-type: none"> ■ Enter Manually Click to manually enter message data in the Enter Value field. A Generate Sample button enables you to automatically generate a sample file for testing. Click Save As to save the sample file. ■ Load From File Click the Browse icon to load message data from a file. The file is added to the messages folder in the Application Navigator.
Duration	Enter the maximum amount of time to wait for the fault message to be delivered from the web service partner.

[Figure 49–14](#) shows this dialog:

Figure 49–14 Emulate Dialog with Emulate Fault Selected



An example of a simulated fault message from a web service partner that you enter manually or load from a file is shown in [Section 49.2.2, "Emulations."](#)

6. Click **OK**.

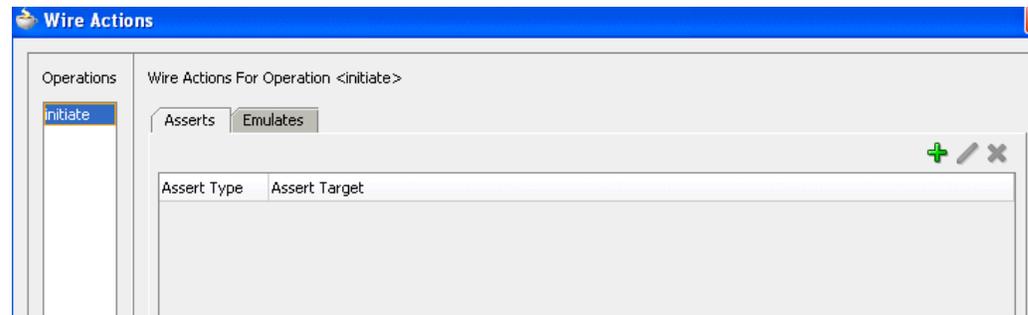
49.4.5 How to Create Assertions

To create assertions:

You perform assertions to verify variable data or process flow. Assertions enable you to validate test data in an entire XML document, a part section of a message, a nonleaf element, or a leaf element as a process is executed. This is done by extracting a value and comparing it to an expected value.

1. Access the Wire Actions dialog by following Step 1 through Step 3 on page 49-10.
2. Click the **Asserts** tab.

[Figure 49–15](#) shows this dialog:

Figure 49–15 Wire Actions Dialog with Asserts Tab Selected

3. Click the **Add** icon.

The Create Assert dialog appears.

4. Select the type of assertion to perform at the top of the dialog, as shown in [Table 49–5](#). If the operation supports only input messages, the **Assert Input** button is enabled. If the operation supports both input and output messages, the **Assert Input** and **Assert Output** buttons are both enabled.

Table 49–5 Assertion Types

Type	Description
Assert Input	Select to create an assertion in the inbound direction.
Assert Output	Select to create an assertion in the outbound direction.
Assert Callback	Select to create an assertion on a callback.
Assert Fault	Select to assert a fault into the application flow.

5. See the section shown in [Table 49–6](#) based on the type of assertion you want to perform.

Table 49–6 Assertion Types

For an Assertion on...	See...
<ul style="list-style-type: none"> ■ A part section of a document ■ A nonleaf element ■ An entire XML document 	Section 49.4.5.1, "Creating Assertions on a Part Section, Nonleaf Element, or Entire XML Document"
A leaf element	Section 49.4.5.2, "Creating Assertions on a Leaf Element"

49.4.5.1 Creating Assertions on a Part Section, Nonleaf Element, or Entire XML Document

To create assertions on a part section, nonleaf element, or entire XML document:

This test compares the values to the expected values.

Note: If the message contains multiple parts (for example, **payload1**, **payload2**, and **payload3**), you must create separate assertions for each part.

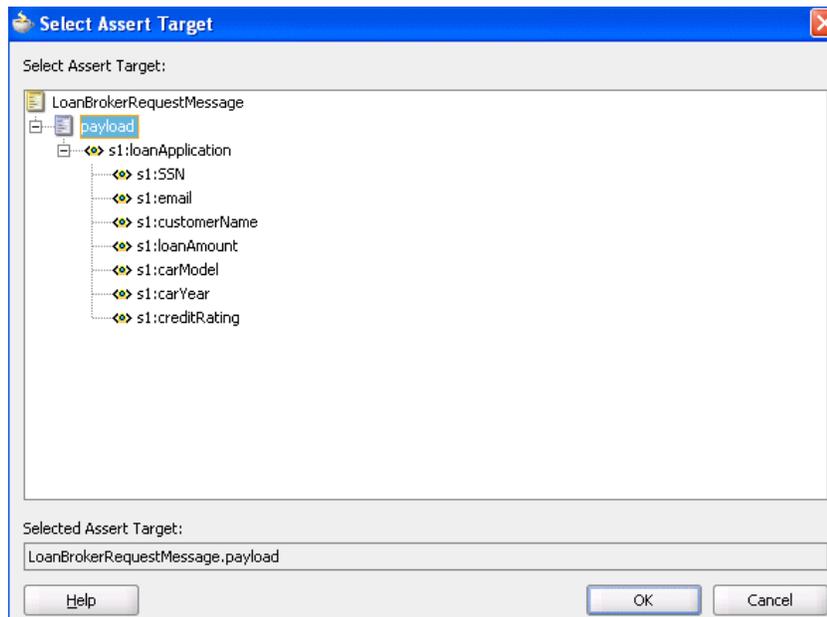
1. Click **Browse** to select the target part section, nonleaf element, or entire XML document to assert.

The Select Assert Target dialog appears.

2. Select a value, and click **OK**. For example, select a variable such as **payload** to perform a part section assertion.

Figure 49–16 shows this dialog. While this example shows how to perform a part section assertion, selecting **LoanBrokerRequestMessage** is an example of an entire XML document assertion and selecting **loanApplication** is an example of a nonleaf assertion.

Figure 49–16 *Select a Part Section of a Message*



The Create Assert dialog refreshes based on your selection of a variable.

3. Enter details in the remaining fields, as shown in Table 49–7:

Table 49–7 *Create Assert Dialog Fields and Values*

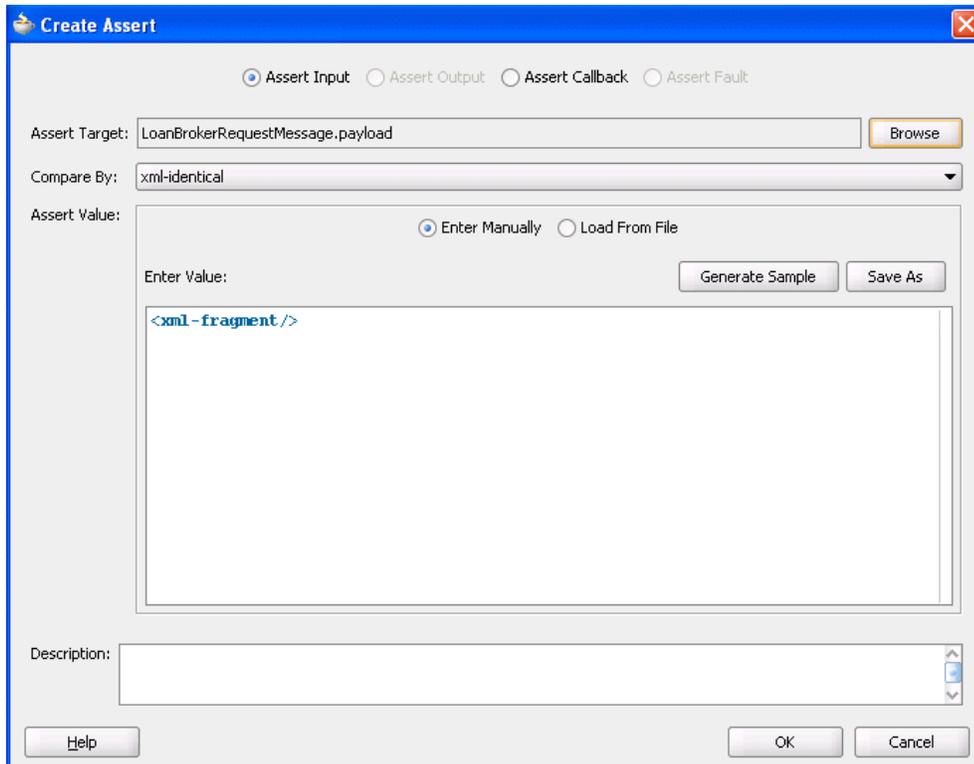
Field	Value
Fault	Select the type of fault to assert (for example, NegativeCredit). This field only displays if you select Assert Fault in Step 4 on page 49-17.
Assert Target	Displays the assert target you selected in Step 2.

Table 49–7 (Cont.) Create Assert Dialog Fields and Values

Field	Value
Compare By	<p>Specify the strictness of the comparison.</p> <ul style="list-style-type: none"> ■ xml-identical: Used when the comparison between the elements and attributes of the XML documents must be exact. If there is any difference between the two XML documents, the comparison fails. For example, the comparison fails if one document uses an element name of <code>purchaseOrder</code>, while the other uses an element name of <code>invoice</code>. The comparison also fails if the child attributes of two elements are the same, but the attributes are ordered differently in each element. ■ xml-similar: Used when the comparison must be similar in content, but does not need to exactly match. For example, the comparison succeeds if both use the same namespace URI, but have different namespace prefixes. The comparison also succeeds if both contain the same element with the same child attributes, but the attributes are ordered differently in each element. <p>In both of these examples, the differences are considered recoverable, and therefore similar.</p> <p>For more information about comparing the contents of XML files, see the XMLUnit web site:</p> <p>http://xmlunit.sourceforge.net/userguide/html/ar01s03.html#The%20Difference%20Engine</p>
Part	Select the message part containing the XML document (for example, payload).
Value	<p>Create an XML document whose content is compared to the assert target content:</p> <ul style="list-style-type: none"> ■ Enter Manually Click to manually enter message data in the Enter Value field. A Generate Sample button enables you to automatically generate a sample file for testing. Click Save As to save the sample file. ■ Load From File Click the Browse icon to load message data from a file. The file is added to the messages folder in the Application Navigator.
Description	Enter an optional description.

Figure 49–17 shows this dialog with **Assert Input** selected:

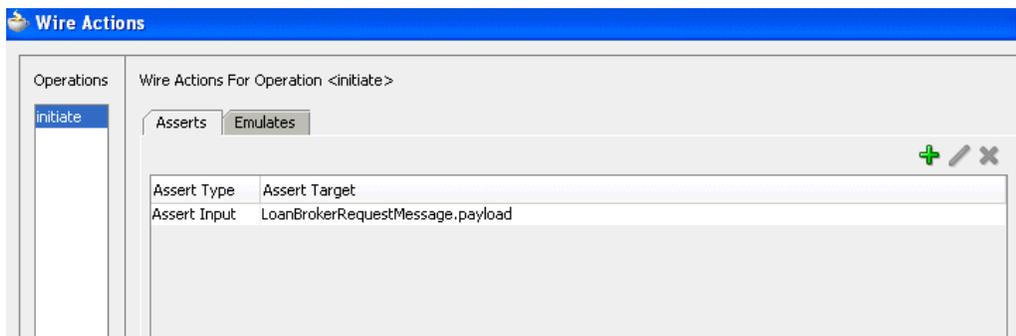
Figure 49–17 Create Assert Dialog with Assert Input Selected



4. Click **OK**.

The Wire Actions dialog shown in [Figure 49–18](#) displays your selection.

Figure 49–18 Wire Actions Dialog with Asserts Tab Selected



5. Click **OK**.

49.4.5.2 Creating Assertions on a Leaf Element

To create assertions on a leaf element:

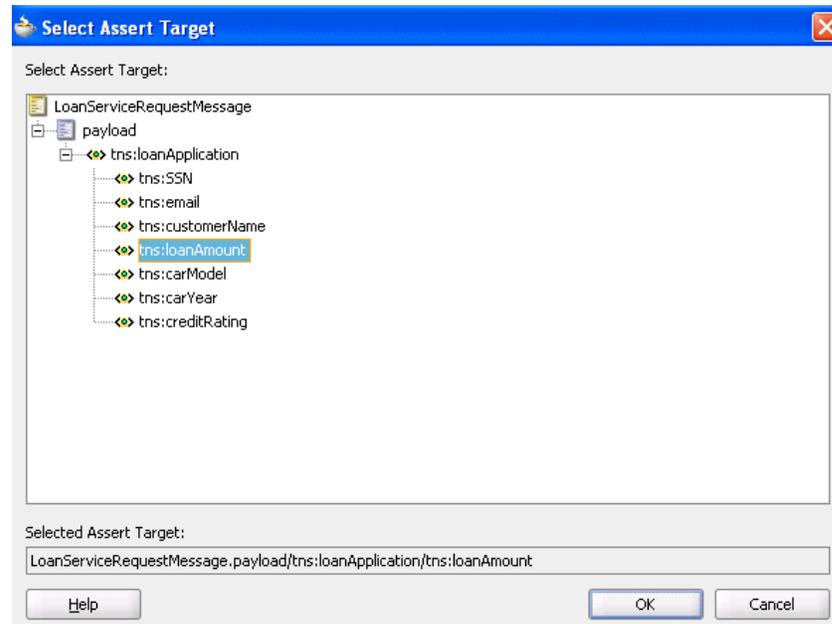
This test compares the value to an expected value.

1. Click **Browse** to select the leaf element to assert.

The Select Assert Target dialog appears.

2. Select a leaf element, and click **OK**. For example, select **loanAmount** to perform an assertion. [Figure 49–19](#) provides details.

Figure 49–19 Selection of a Leaf Element



The Create Assert dialog refreshes based on your selection of an entire XML document.

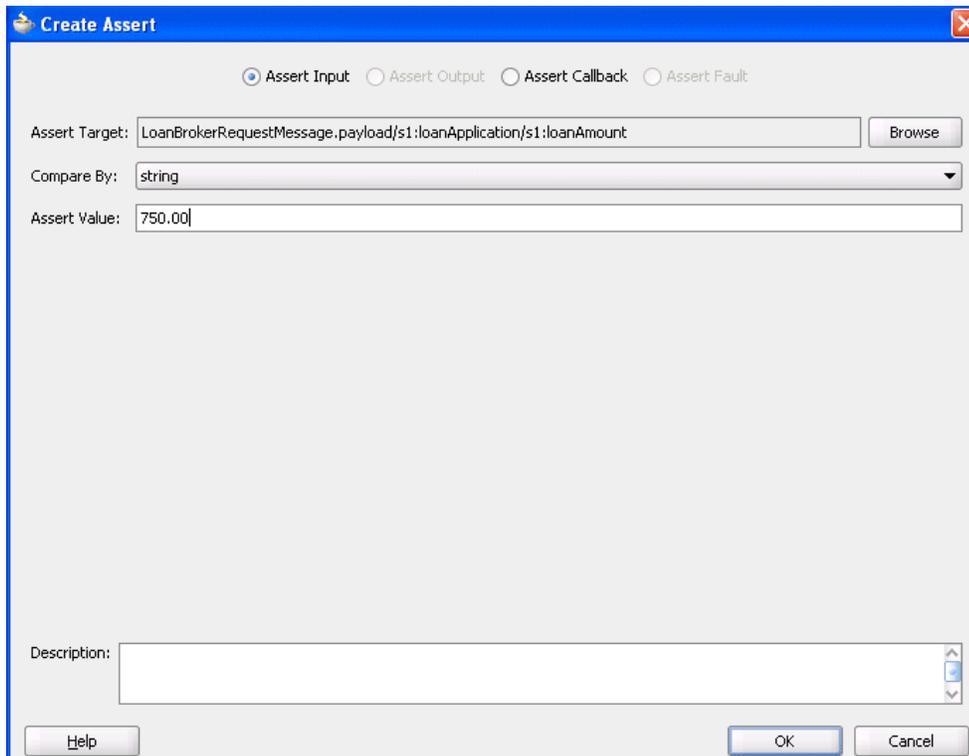
3. Enter details in the remaining fields, as shown in [Table 49–8](#):

Table 49–8 Create Assert Dialog Fields and Values

Field	Value
Fault	Select the type of fault to assert (for example, NegativeCredit). This field only displays if you select Assert Fault in Step 4 on page 49-17.
Callback Operation	Select the type of callback to assert (for example, onResult). This field only displays if you select Assert Callback in Step 4 on page 49-17.
Assert Target	Displays the variable assert target you selected in Step 2.
Compare By	Select the type of comparison: <ul style="list-style-type: none"> ■ string: Compares string values ■ number: Compares numerical values ■ pattern-match: Compares a regular expression pattern (for example, <code>[0-9]*</code>). Java Development Kit (JDK) regular expression (regexp) constructs are supported. For example, entering a pattern of <code>ab[0-9]*cd</code> means that a value of <code>ab123cd</code> or <code>ab456cd</code> is correct. An asterisk (*) indicates any number of occurrences.
Assert Value	Enter the value you are expecting. This value is compared to the value for the assert target.
Description	Enter an optional description.

[Figure 49–20](#) shows this dialog with **Assert Input** selected:

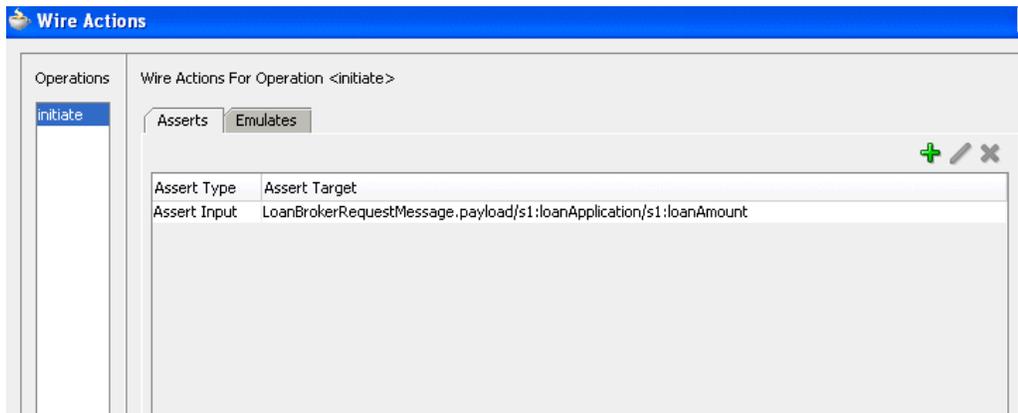
Figure 49–20 Create Assert Dialog



4. Click **OK**.

The Wire Actions dialog shown in [Figure 49–18](#) displays your selection.

Figure 49–21 Wire Actions Dialog with Asserts Tab Selected



49.4.6 What You May Need to Know About Assertions

When a test is executed, and the response type returned is different from the type expected, the assertion is skipped. For example, you are expecting a fault (`RemoteFault`) to be returned for a specific message, but a response (`BpelResponseMessage`) is instead returned.

As a best practice, always assert and emulate the expected behavior.

49.5 Deploying and Running a Test Suite

After creating a test suite of test cases, you deploy the suite as part of a SOA composite application. You then run the test suites from Oracle Enterprise Manager Fusion Middleware Control Console.

See [Section 43.2.1, "How to Deploy a Single SOA Composite"](#) for instructions on deploying a SOA composite application from Oracle JDeveloper. See *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite* for instructions on deploying a SOA composite application and running a test suite from Oracle Enterprise Manager Fusion Middleware Control Console.

Managing Policies

This chapter describes how to manage policies during design-time in SOA composite applications.

This chapter includes the following sections:

- [Section 50.1, "Introduction to Policies"](#)
- [Section 50.2, "Attaching Policies to Binding Components and Service Components"](#)

50.1 Introduction to Policies

Oracle Fusion Middleware uses a policy-based model to manage and secure Web services across an organization. Policies apply security to the delivery of messages. Policies can be managed by both developers in a design time environment and system administrators in a runtime environment.

Policies are comprised of one or more assertions. A policy assertion is the smallest unit of a policy that performs a specific action. Policy assertions are executed on the request message and the response message, and the same set of assertions is executed on both types of messages. The assertions are executed in the order in which they appear in the policy.

[Table 50–1](#) describes the supported policy categories.

Table 50–1 Supported Policy Categories

Category	Description
Message Transmission Optimization Mechanism (MTOM)	Ensures that attachments are in MTOM format. This format enables binary data to be sent to and from web services. This reduces the transmission size on the wire.
Reliability	Supports the WS-Reliable Messaging protocol. This guarantees the end-to-end delivery of messages.
Addressing	Verifies that simple object access protocol (SOAP) messages include WS-Addressing headers in conformance with the WS-Addressing specification. Transport-level data is included in the XML message rather than relying on the network-level transport to convey this information.
Security	Implements the WS-Security 1.0 and 1.1 standards. They enforce authentication and authorization of users, identity propagation, and message protection (message integrity and message confidentiality).
Management	Logs request, response, and fault messages to a message log. Management policies can also include custom policies.

Within each category there are one or more policy types that you can attach. For example, if you select the reliability category, the following types are available for selection:

- oracle/wsrml0_policy
Supports version 1.0 of the Web Services Reliable Messaging protocol.
- oracle/wsrml1_policy
Supports version 1.1 of the Web Services Reliable Messaging protocol.

For more information about available policies and details about which ones to use in your environment, see *Oracle Fusion Middleware Security and Administrator's Guide for Web Services*.

50.2 Attaching Policies to Binding Components and Service Components

You can attach or detach policies to and from service binding components, service components, and reference binding components in a SOA composite application. Use Oracle JDeveloper to attach policies for testing security in a design-time environment. When your application is ready for deployment to a production environment, you can attach or detach runtime policies in Oracle Enterprise Manager Fusion Middleware Control Console.

For more information about runtime management of policies, see *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite*.

50.2.1 How to Attach Policies to Binding Components and Service Components

To attach a policy to a service or reference binding component:

1. In the SOA Composite Editor, right-click a service binding component or reference binding component.
2. Select **Configure WS-Policies**.

Depending upon the interface definition of your SOA composite application, you may be prompted with an additional menu of options.

- If the selected service or reference is interfacing with a synchronous BPEL process or Oracle Mediator service component, a single policy is used for both request and response messages. The Configure SOA WS Policies dialog immediately appears. Go to Step 4.
- If the service or reference is interfacing with an asynchronous BPEL process or Oracle Mediator service component, the policies must be configured separately for request and response messages. The policy at the callback is used for the response sent from service to client. An additional menu is displayed. Go to Step 3.

3. Select the type of binding to use:

- **For Request:**

Select the request binding for the service component with which to bind. You can only select a single request binding. This action enables communication between the binding component and the service component.

When request binding is configured for a service in the **Exposed Services** swimlane, the service acts as the server. When request binding is configured

for a reference in the **External References** swimlane, the reference acts as the client.

- **For Callback:** (only for interactions with asynchronous processes)

Select the callback binding for the service component with which to bind. This action enables message communication between the binding component and the service component. You can only select a single callback binding.

When callback binding is configured for a service in the **Exposed Services** swimlane, the service acts as the client. When callback binding is configured for a reference in the **External References** swimlane, the reference acts as the server.

The Configure SOA WS Policies dialog shown in [Figure 50–1](#) appears. For this example, the **For Request** option was selected for a service binding component. The same types of policy categories are also available if you select **For Callback**.

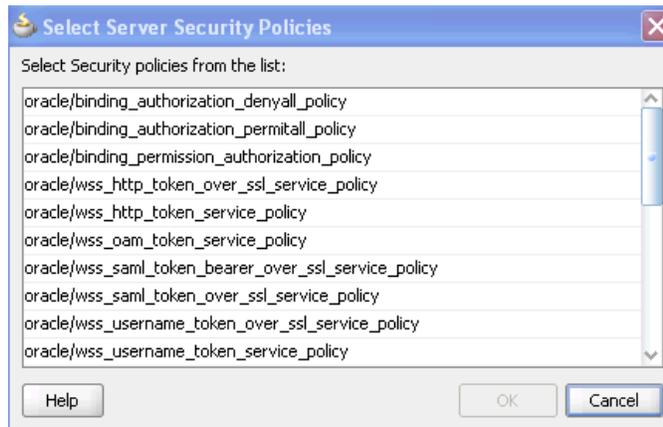
Figure 50–1 Configure SOA WS Policies Dialog



4. Click the **Add** icon to display the following categories of polices. For this example, **Security** is selected for attachment.
 - **MTOM**
 - **Reliability**
 - **Addressing**
 - **Security**
 - **Management**

The dialog shown in [Figure 50–2](#) is displayed.

Figure 50–2 Security Policies



5. Place your cursor over a policy name to display a description of policy capabilities.
6. Select the type of policy to attach.
7. Click **OK**.

You are returned to the Configure SOA WS Policies dialog shown in [Figure 50–3](#). The attached security policy displays in the **Security** section.

Figure 50–3 Attached Security Policy

8. If necessary, add additional policies.

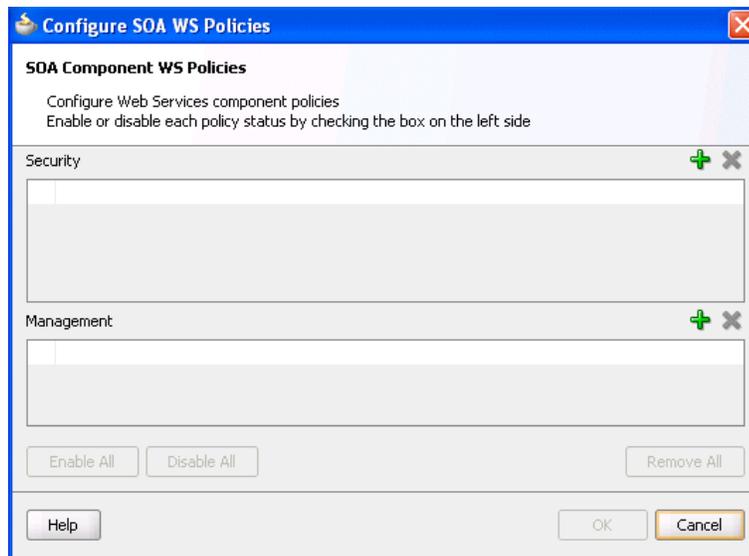
You can temporarily disable a policy by deselecting the checkbox to the left of the name of the attached policy. This action does *not* detach the policy.

9. To detach a policy, click the **Delete** icon.
10. When complete, click **OK** on the Configure SOA WS Policies dialog.
You are returned to the SOA Composite Editor.

To attach a policy to a service component:

1. Right-click a service component.
2. Select **Configure Component WS Policies**.

The Configure SOA WS Policies dialog shown in [Figure 50–4](#) appears.

Figure 50–4 Configure SOA WS Policies Dialog

3. Click the **Add** icon for the type of policy to attach.
 - **Security**
 - **Management**The dialog for your selection appears.
4. Select the type of policy to attach.
5. Click **OK**.
6. If necessary, add additional policies.
7. When complete, click **OK** on the Configure SOA WS Policies dialog.

For information about attaching policies during runtime in Oracle Enterprise Manager Fusion Middleware Control Console, see *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite*.

Defining Composite Sensors

This chapter describes how to define composite sensors in an SOA composite application.

This chapter includes the following sections:

- [Section 51.1, "Introduction to Composite Sensors"](#)
- [Section 51.2, "Adding Composite Sensors"](#)
- [Section 51.3, "Monitoring Composite Sensor Data During Runtime"](#)

Note: Only the database sensor action is supported for this release. For more information about sensor actions, see [Chapter 17, "Using Oracle BPEL Process Manager Sensors."](#)

51.1 Introduction to Composite Sensors

Composite sensors provide a method for implementing trackable fields on messages. Composite sensors enable you to perform the following tasks:

- Monitor incoming and outgoing messages.
- Specify composite sensor details in the search utility of the Instances page of an SOA composite application in Oracle Enterprise Manager Fusion Middleware Control Console. This action enables you to locate a particular instance.

You define composite sensors on service and reference binding components in Oracle JDeveloper. This functionality is similar to variable sensors in BPEL processes. During runtime, composite sensor data is persisted in the database.

51.1.1 Restrictions on Use of Composite Sensors

Note the following restrictions on the use of composite sensors:

- Functions can only be used with the payload. For example, XPath functions such as `concat()` and others cannot be used with properties.
- Any composite sensor that uses expressions always captures values as strings. This action makes the search possible only with the `like` comparison operator. Also, even if the value is a number, you cannot use other logical operators such as `<`, `>`, `=`, and any combination of these.
- Composite sensors only support the predefined **DBSensorAction**.
- Header-based sensors are only supported for web service bindings.

- Sensors for Oracle B2B, service data objects (SDOs), web services invocation framework (WSIF), and Oracle Business Activity Monitoring bindings are not supported.
- Sensor values can only be one of the following types.
 1. The following scalar types:
 - STRING
 - NUMBER
 - DATE
 - DATE_TIME
 2. Complex XML elements
- When creating an XPath expression for filtering, all functions that return a node set must be explicitly cast as a string:


```

      xpath20:upper-case(string($in.request/inp1:updateOrderStatus/inp1:orderStatus)
      ) = "PENDING"
      
```

51.2 Adding Composite Sensors

You add sensors to service or reference binding components of an SOA composite application in the SOA Composite Editor.

51.2.1 How to Add Composite Sensors

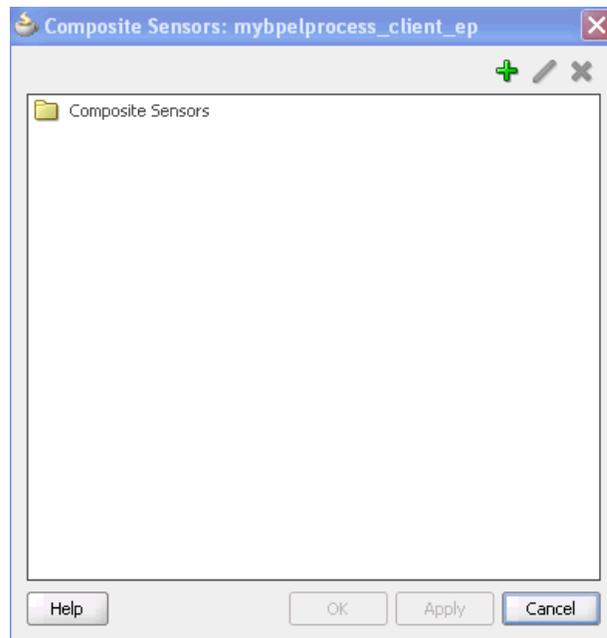
To add composite sensors:

1. Use one of the following options to add a composite sensor in the SOA Composite Editor.
 - a. Right-click a specific service or reference binding component to which to add a composite sensor, and select **Composite Sensor**.
 - b. Click the **Composite Sensor** icon above the SOA Composite Editor.

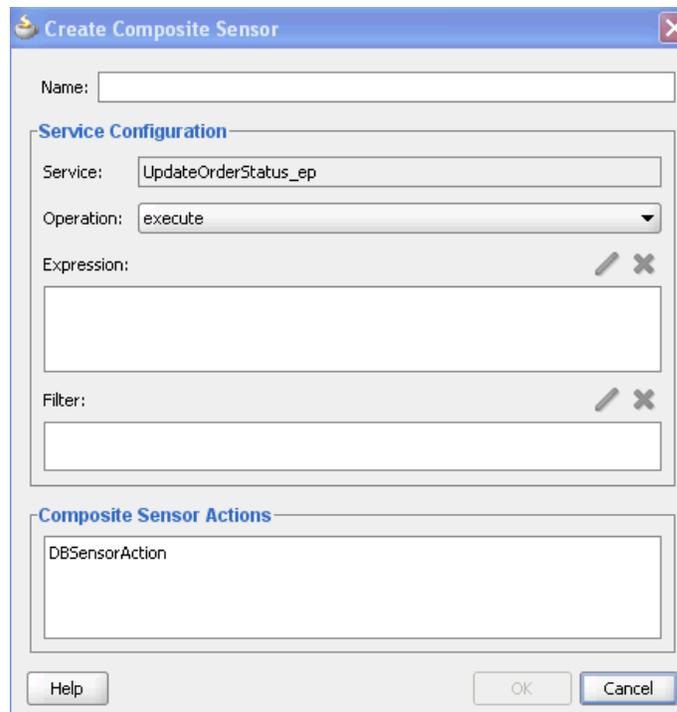
Figure 51–1



The Composite Sensors dialog appears, as shown in [Figure 51–2](#).

Figure 51–2 Composite Sensor Dialog

2. Select the **Composite Sensors** folder, then click the **Add** icon.
The Create Composite Sensor dialog appears, as shown in [Figure 51–3](#).

Figure 51–3 Create Composite Sensor Dialog

3. Enter the details shown in [Table 51–1](#).

Table 51–1 Create Composite Sensor Dialog

Name	Description
Name	Enter a name for the composite sensor. You must enter a name to enable the Edit icon of the Expression field.
Service	<p>Displays the name of the service. This field only displays if you are creating a composite sensor for a service binding component. This field cannot be edited.</p> <p>Service sensors monitor the messages that the service receives from the external world or from another composite application.</p>
Reference	<p>Displays the name of the reference. This field only displays if you are creating a composite sensor for a reference binding component. This field cannot be edited.</p> <p>Reference sensors monitor the messages that the reference sends to the external world or to another composite application.</p>
Operation	Select the operation for the port type of the service or reference.
Expression	<p>Click the Edit icon to invoke a dropdown list for selecting the type of expression to create:</p> <ul style="list-style-type: none"> ■ Variables: Select to create an expression value for a variable. See Section 51.2.2, "Adding a Variable" for instructions. ■ Expression: Select to invoke the Expression Builder dialog for creating an XPath expression. See Section 51.2.3, "Adding an Expression" for instructions. ■ Properties: Select to create an expression value for a normalized message header property. These are the same properties that display under the Properties tab of the invoke activity, receive activity, reply activity, and OnMessage branch of a pick activity. See Section 51.2.4, "Adding a Property" for instructions.
Filter	<p>Click the Edit icon to invoke the Expression Builder dialog to create an XPath filter for the expression. You must first create an expression to enable this field.</p> <p>For example, you may create an expression for tracking purchase order amounts over 10,000:</p> <pre>\$in.inDict/tns:inDict/ns2:KeyValueOfstringstring/ns2:Value > 10000.00</pre>
Composite Sensor Actions	Displays the supported sensor action (DBSensorAction). This feature enables runtime sensor data to be stored in the database. For this release, only this sensor action type is supported for composite sensors. This field cannot be edited.

4. Click **Apply** to add more composite sensors.
5. Click **OK** when complete.

A **sensor** icon displays on the service or reference binding component.

Figure 51–4 Sensor Icon



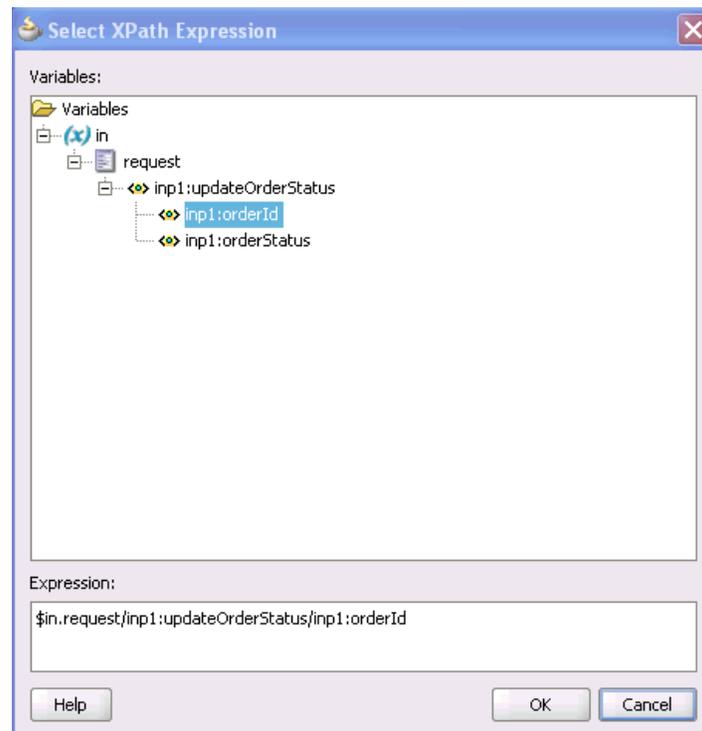
51.2.2 Adding a Variable

The Select XPath Expression dialog shown in [Figure 51–5](#) enables you to select an element for tracking.

To add a variable:

1. Expand the tree and select the element to track.

Figure 51–5 Variables



2. Click OK when complete.

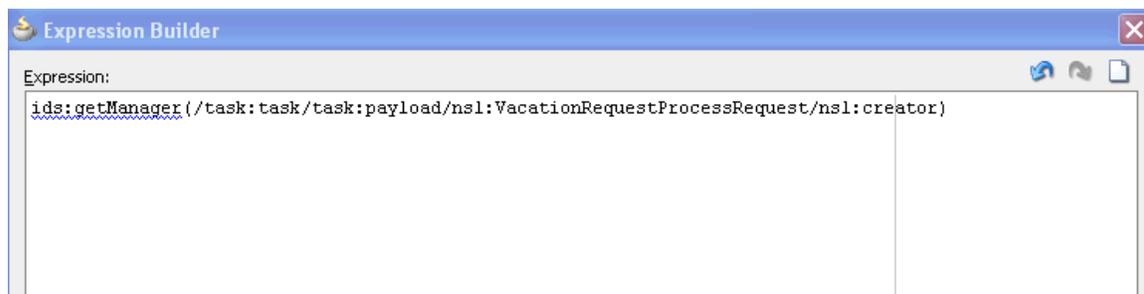
51.2.3 Adding an Expression

The Select Properties shown in [Figure 51–6](#) enables you to create an expression for tracking.

To add an expression:

1. Build an XPath expression of an element to track.

Figure 51–6 Expression



2. Click **OK** when complete.

51.2.4 Adding a Property

The Select Property shown in [Figure 51–7](#) enables you to select a normalized message header property for tracking.

To add a property:

1. Select a normalized message header property to track.

Figure 51–7 Properties

Selected Operation:
execute

Select Message Type:
\$in

Select Property From List:

- jca.mq.Inbound.MQMD.Report.Generate.MsgId
- jca.mq.MQMD.AccountingToken
- jca.mq.MQMD.ApplIdentityData
- jca.mq.MQMD.ApplOriginData
- jca.mq.MQMD.BackoutCount
- jca.mq.MQMD.CodedCharSetId
- jca.mq.MQMD.CorrelId
- jca.mq.MQMD.Encoding.Decimal
- jca.mq.MQMD.Encoding.Float
- jca.mq.MQMD.Encoding.Integer
- jca.mq.MQMD.Expiry
- jca.mq.MQMD.Feedback
- jca.mq.MQMD.Feedback.ApplicationDefined
- jca.mq.MQMD.Format

Selected Property:

Help OK Cancel

2. Click **OK** when complete.

51.3 Monitoring Composite Sensor Data During Runtime

During runtime, composite sensor data can be monitoring in Oracle Enterprise Manager Fusion Middleware Control Console:

- Composite sensor data displays in the flow trace of an SOA composite application.
- Composite sensor data can be searched for in the Instances page of an SOA composite application.

For more information, see *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite*.

Using Service Data Objects and Enterprise JavaBeans

This chapter describes how Enterprise JavaBeans and SOA composite applications interact by passing service data object (SDO) parameters.

This chapter includes the following sections:

- [Section 52.1, "Introduction to SDO and Enterprise JavaBeans Binding"](#)
- [Section 52.2, "Designing an Enterprise JavaBeans Application"](#)
- [Section 52.3, "Creating an Enterprise JavaBeans Adapter Service in Oracle JDeveloper"](#)
- [Section 52.4, "Designing an Enterprise JavaBeans Client to Invoke Oracle SOA Suite"](#)
- [Section 52.5, "Specifying Enterprise JavaBeans Roles"](#)
- [Section 52.6, "Configuring JNDI Access"](#)

52.1 Introduction to SDO and Enterprise JavaBeans Binding

SDOs enable you to modify business data regardless of how it is physically accessed. Knowledge is not required about how to access a particular back-end data source to use SDO in an SOA composite application. Consequently, you can use static or dynamic programming styles and obtain connected and disconnected access.

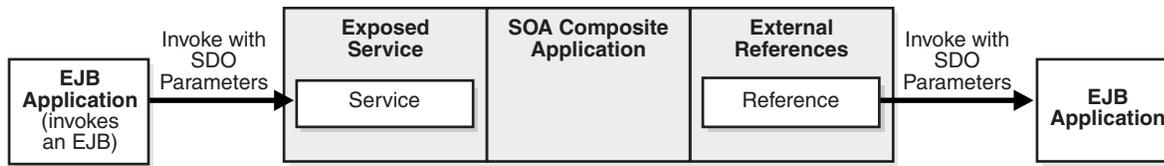
Enterprise JavaBeans are server-side domain objects that fit into a standard component-based architecture for building enterprise applications with Java. These objects become distributed, transactional, and secure components.

Oracle SOA Suite interfaces are described by the Web Services Description Language (WSDL) file. Enterprise JavaBeans interfaces are described by Java interfaces. Invocations between the two are made possible in Oracle SOA Suite by an Enterprise JavaBeans Java interface that corresponds to an Oracle SOA Suite WSDL interface.

Through this interface, Oracle SOA Suite provides support for the following:

- Invoking Enterprise JavaBeans with SDO parameters through an Enterprise JavaBeans adapter reference. In this scenario, a SOA composite application passes SDO parameters to an external Enterprise JavaBeans application.
- Invoking an Enterprise JavaBeans adapter service through Enterprise JavaBeans with SDO parameters. In this scenario, an Enterprise JavaBeans application passes SDO parameters into a SOA composite application.

[Figure 52-1](#) provides an overview.

Figure 52–1 SDO and Enterprise JavaBeans Binding Integration

52.2 Designing an Enterprise JavaBeans Application

This section provides a high-level overview of the steps for designing an Enterprise JavaBeans application. For more information, see the following documentation:

- *Oracle Fusion Middleware Programming Enterprise JavaBeans, Version 3.0 for Oracle WebLogic Server*
- *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*
- Oracle JDeveloper online help table of contents for the following topics:
 - Enterprise JavaBeans
 - SDO for Enterprise JavaBeans/JPA

Access the help by selecting **Help > Table of Contents** in Oracle JDeveloper.

52.2.1 How to Create SDO Objects Using the SDO Compiler

Select one of the following options for creating SDO objects:

- EclipseLink is an open source, object-relational mapping package for Java developers. EclipseLink provides a framework for storing Java objects in a relational database or converting Java objects to XML documents.

Use EclipseLink to create SDO objects. For instructions on installing, configuring, and using EclipseLink to create SDO objects, visit the following URL:

http://wiki.eclipse.org/EclipseLink/Installing_and_Configuring_EclipseLink
- Oracle JDeveloper enables you to create an SDO service interface for JPA entities. While this feature is more tailored for use with the Oracle Application Development Framework (ADF) service binding in a SOA composite application, you can also use this feature with the Enterprise JavaBeans service binding in SOA composite applications. The SDO service interface feature generates the necessary WSDL and XSD files. If you use this feature, you must perform the following tasks to work with the Enterprise JavaBeans service binding:
 - Browse for and select this WSDL file in the SOA Resource Browser dialog, which is accessible from the **WSDL URL** field of the Create EJB Service dialog (described in [Section 52.3, "Creating an Enterprise JavaBeans Adapter Service in Oracle JDeveloper"](#)).
 - Add the **BC4J Service Runtime** library to the SOA project. To add this library, double-click the project and select **Libraries and Classpath** to add the library in the Project Properties dialog. You are now ready to design the logic.

For more information, see the SDO for Enterprise JavaBeans/JPA topic in the Oracle JDeveloper online help (this includes instructions on how create to an SDO service interface).

52.2.2 How to Create a Session Bean and Import the SDO Objects

To create a session bean and import the SDO objects:

1. Create a simple session bean with the Create Session Bean wizard. For details on using this wizard, see the Creating a Session Bean topic in the Oracle JDeveloper online help.
2. Import the SDO objects into your project through the Project Properties dialog.
3. Add logic and necessary import and library files. In particular, you must import the `Commonj.sdo.jar` file. JAR files can be added in the Libraries and Classpath dialog. This dialog is accessible by double-clicking the project and selecting **Libraries and Classpath** in the Project Properties dialog. You are now ready to design the logic.
4. Expose the method to the remote interface.

52.2.3 How to Create a Profile and an EAR File

To create a profile and an EAR file:

1. Create an Enterprise JavaBeans JAR profile in the Project Properties dialog.
2. Create an application level EAR file in the Application Properties dialog.

52.2.4 How to Define the SDO Types with an Enterprise JavaBeans Bean

An Enterprise JavaBeans bean must define the SDO types. [Example 52–1](#) provides details.

Caution: Where to call `define` can be nontrivial. You must force the types to be defined before remote method invocation (RMI) marshalling must occur and in the right helper context. The EclipseLink SDO implementation indexes the helper instance with the application name or class loader.

When you invoke the Enterprise JavaBeans method, an application name is available to the EclipseLink SDO runtime. The EclipseLink SDO looks up the context using the application name as the key. Ensure that the types are defined when the application name is visible. When an Enterprise JavaBeans static block is initialized, the application name is not created. Therefore, putting the `define` in the static block does not work if you are using the default application name-based context. One way to get the application name initialized is to allocate more than two instance beans using the `weblogic-ejb-jar.xml` file.

Example 52–1 Definition of SDO Types

```
InputStreamReader reader = new InputStreamReader(url.openStream());
StreamSource source = new StreamSource(reader);
List<SDOType> list = ((SDOXSDHelper) XSDHelper.INSTANCE).define(source, null);
```

The `weblogic-ejb-jar.xml` file is the descriptor file that must be added in the deployment jar. The `weblogic-ejb-jar.xml` file is automatically created when you create a session bean. This file must be modified by adding the following entries.

Example 52–2 weblogic-ejb-jar.xml File

```
<?xml version = '1.0' encoding = 'windows-1252'?>
<weblogic-ejb-jar xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.bea.com/ns/weblogic/weblogic-ejb-jar
http://www.bea.com/ns/weblogic/weblogic-ejb-jar/1.0/weblogic-ejb-jar.xsd"
xmlns="http://www.bea.com/ns/weblogic/weblogic-ejb-jar">

    <weblogic-enterprise-bean>
        <ejb-name>HelloEJB</ejb-name>
        <stateless-session-descriptor>
            <pool>
                <initial-beans-in-free-pool>2</initial-beans-in-free-pool>
            </pool>
        </stateless-session-descriptor>
    </weblogic-enterprise-bean>

</weblogic-ejb-jar>
```

Figure 52–2 provides a code example of a session bean with SDO logic defined:

Figure 52–2 Session Bean with Defined SDO Logic

```
package sdo.ejb.employee;

import ...;

@Stateless(name = "SessionEJB", mappedName = "sdo-ejb-SessionEJB")
@Remote
@WebService(portName = "SessionEJBBeanServicePort", endpointInterface = "sdo.ejb.employee.SessionEJB")
public class SessionEJBBean implements SessionEJB {
    public SessionEJBBean() {
        try {
            defineSchema("/", "employee.xsd");
            System.out.println("Successfully initialized!");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public EmployeeResponse getEmployeeInfo(Employee emp){
        System.out.println("Emp SSN -->" +emp.getEmp().getSSN());
        EmployeeDetails empDetails = emp.getEmp();
        EmployeeResponse response = (EmployeeResponse) DataFactory.INSTANCE.create(EmployeeResponse.class);
        empDetails.setEmployeeType("Full Time");
        empDetails.setSSN(emp.getEmp().getSSN());
        response.setResult(empDetails);
        return response;
    }

    private static List defineSchema(String resourceLoc, String resourceName) throws IOException {

        ClassLoader cl = Thread.currentThread().getContextClassLoader();
        URL url = cl.getResource(resourceLoc + resourceName);
        if (url == null)
            throw new IOException("Can't read " + resourceLoc + resourceName);

        InputStreamReader reader = new InputStreamReader(url.openStream());
        StreamSource source = new StreamSource(reader);
        return ((SDOXSDHelper) XSDHelper.INSTANCE).define(source, null);
    }
}
```

52.2.5 How to Use Web Service Annotations

In order to generate the WSDL file, the Enterprise JavaBeans interface must use the following web service annotations. Use of these annotations is described in JSR 224: Java API for XML-Based Web Services (JAX-WS) 2.0. Visit the following URL for details:

<http://www.jcp.org/en/jsr/detail?id=224>

In addition, only a document/literal WSDL is currently supported by the Enterprise JavaBeans binding layer.

Table 52-1 describes the annotations to use.

Table 52-1 Annotations

Name	Description
<code>@javax.jws.WebResult;</code> <code>@javax.jws.WebParam;</code>	Customize the mapping of an individual parameter to a web service message part and XML element. Both annotations are used to map SDO parameters to the correct XML element from the normalized message payload.
<code>@javax.jws.Oneway;</code>	Denote a method as a web service one-way operation that has only an input message and no output message. The Enterprise JavaBeans binding component does not expect any reply in this case.
<code>@javax.xml.ws.RequestWrapper;</code>	Tell the Enterprise JavaBeans binding components whether the deserialized object must be unwrapped or whether a wrapper must be created before serialization.
<code>@javax.xml.ws.ResponseWrapper;</code>	An Enterprise JavaBeans interface can be generated from an existing WSDL or obtained by some other means. If the WSDL does not exist, it can be generated.
<code>@javax.xml.ws.WebFault;</code>	Map WSDL faults to Java exceptions. This annotation captures the fault element name used when marshalling the JAXB type generated from the global element referenced by the WSDL fault message.
<code>@oracle.webservices.PortableWebService</code>	Specify the <code>targetNamespace</code> and <code>serviceName</code> used for the WSDL. For example: <pre>@PortableWebService(targetNamespace = "http://hello.demo.oracle/", serviceName = "HelloService")</pre> The <code>serviceName</code> is used as the WSDL file name. If it is not specified in the annotations, the SEI class name is used instead.
Add appropriate method parameter annotations	Add to control how message elements and types are mapped to the WSDL. For example, if your interface is in <code>doc/lit/bare</code> style, add the following annotations to the methods. <pre>@WebMethod @SOAPBinding(parameterStyle = SOAPBinding.ParameterStyle.BARE)</pre>
<code>@SDODatabinding</code>	Add to the interface class to use the existing schema instead of a generated one. For example: <pre>@SDODatabinding(schemaLocation = "etc/HelloService.xsd")</pre>

Example 52-3 provides an example of an Enterprise JavaBeans interface with annotations.

Example 52–3 Enterprise JavaBeans Interface with Annotations

```

@Remote
@PortableWebService(targetNamespace = "http://www.example.org/customer-example",
    serviceName = "CustomerSessionEJBService")
@SDODataBinding(schemaLocation = "customer.xsd")
public interface CustomerSessionEJB {
    @WebMethod(operationName="createCustomer")
    @SOAPBinding(parameterStyle = SOAPBinding.ParameterStyle.BARE)
    @WebResult(targetNamespace = "http://www.example.org/customer-example",
        partName = "parameters", name = "customer")
    CustomerType createCustomer();
    @WebMethod(operationName="addPhoneNumber")
    @SOAPBinding(parameterStyle = SOAPBinding.ParameterStyle.BARE)
    @WebResult(targetNamespace = "http://www.example.org/customer-example",
        partName = "parameters", name = "customer")
    CustomerType addPhoneNumber(@WebParam(targetNamespace =
        "http://www.example.org/customer-example", partName = "parameters", name =
        "phone-number") PhoneNumber phNumber);
}

```

52.2.6 How to Deploy the Enterprise JavaBeans EAR File**To deploy the EAR file from Oracle JDeveloper:**

1. Select the Application context menu to the right of the application name.
2. Select **Deploy** and deploy the EAR file to a previously created application server connection.

52.3 Creating an Enterprise JavaBeans Adapter Service in Oracle JDeveloper

This section describes how to create an Enterprise JavaBeans adapter reference or Enterprise JavaBeans adapter service in Oracle JDeveloper. This adapter service enables the Enterprise JavaBeans application to communicate with Oracle SOA Suite and Oracle SOA Suite to communicate with remote Enterprise JavaBeans.

52.3.1 Invoking SDO-based Enterprise JavaBeans from SOA Composite Applications

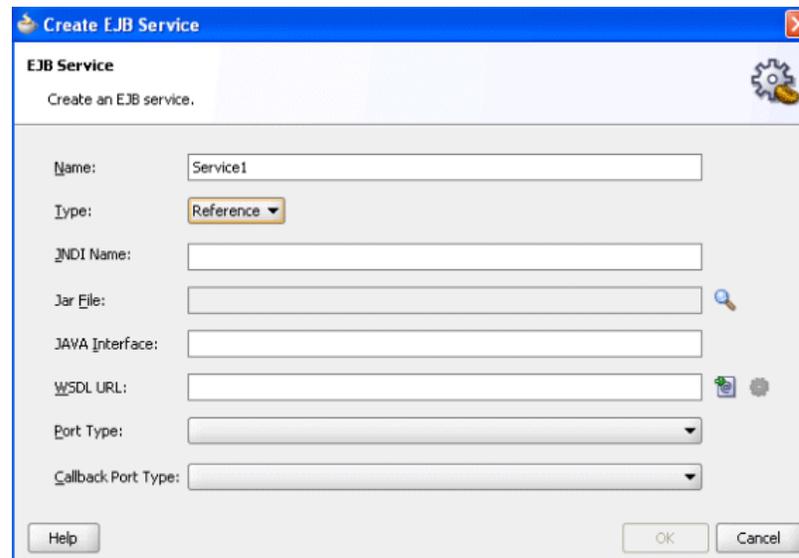
You can invoke Enterprise JavaBeans with SDO parameters from the Enterprise JavaBeans adapter reference in SOA composite applications.

52.3.1.1 How to Invoke SDO-based Enterprise JavaBeans from SOA Composite Applications**To invoke SDO-based Enterprise JavaBeans from SOA composite applications:**

1. Drag an **EJB Adapter Service** icon into the **External References** swimlane.

The Create EJB Service dialog appears, as shown in [Figure 52–3](#).

Figure 52–3 Create EJB Service in External References Swimlane



2. Enter the details shown in [Table 52–2](#):

Table 52–2 Create EJB Service Dialog

Field	Value
Name	Accept the default value or enter a different name.
Type	Displays Reference if you dragged this icon into the External References swimlane.
JNDI Name	Enter the JNDI name of your Enterprise JavaBeans.
Jar File	Click the Search icon to select the Enterprise JavaBeans JAR file created in the previous section. The SOA Resource Browser dialog searches for and displays .jar files starting in the SCA-INF/lib subdirectory of the current project directory. The JAR file includes the interface class and any supporting classes. Note: If you select a JAR file outside of the current project, Oracle JDeveloper creates a copy of the JAR file in the SCA-INF/lib directory of the current project. When prompted, click OK to accept.
Java Interface	Enter the fully qualified Java class name of the previously created Enterprise JavaBeans interface. This class must exist in the selected JAR file. If a JAR file is not specified, it is assumed that the class is in the /SCA-INF/classes subdirectory of the current project directory.
WSDL URL	Note: Ensure that you have created the annotations for the Enterprise JavaBeans interface before generating the WSDL file, as described in Section 52.2.5, "How to Use Web Service Annotations." Click the second icon to the right to generate a WSDL file that represents the Enterprise JavaBeans interface. If you created SDO objects through Oracle JDeveloper, as described in Section 52.2.1, "How to Create SDO Objects Using the SDO Compiler," ensure that you select the WSDL file that was automatically generated with this option.
Port Type	Select the port type.
Callback Port Type	Select the callback port type (for asynchronous services).

3. Click OK.

52.3.2 Invoking SOA Composite Applications from Enterprise JavaBeans using SDO Parameters

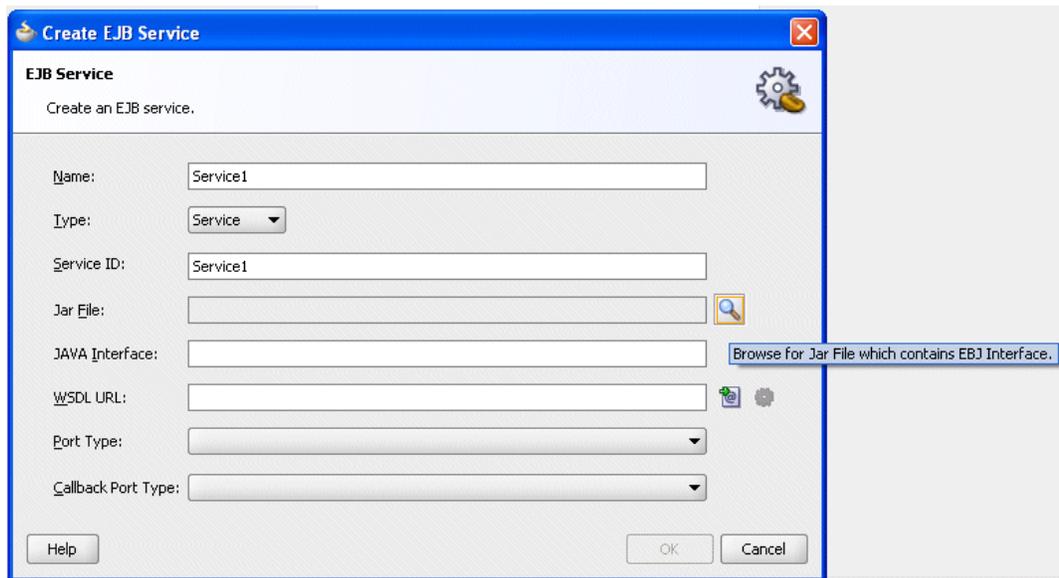
You can invoke an Enterprise JavaBeans adapter service in SOA composite applications from Enterprise JavaBeans using SDO parameters.

52.3.2.1 How to Invoke SOA Composite Applications from Enterprise JavaBeans using SDO Parameters

To invoke SOA composite applications from Enterprise JavaBeans using SDO parameters:

1. Drag an **EJB Adapter Service** icon into the **Exposed Services** swimlane.
The Create EJB Service dialog appears, as shown in [Figure 52-4](#).

Figure 52-4 Create EJB Service in Exposed Services Swimlane



2. Enter the details shown in [Table 52-3](#):

Table 52-3 Create EJB Service in Exposed Services Swimlane

Field	Value
Name	Accept the default value or enter a different name.
Type	Displays Service if you dragged this icon into the Exposed Services swimlane.
Service ID	Accept the default value or enter a different name. The service ID is used as a token to uniquely identify the composite service entry from the Enterprise JavaBeans application. If multiple versions of the same composite are deployed, only the default version is used when the invocation arrives. Different composites trying to use the same service ID receive an error during deployment.

Table 52–3 (Cont.) Create EJB Service in Exposed Services Swimlane

Field	Value
Jar File	Click the Search icon to select the Enterprise JavaBeans JAR file created in the previous section. The SOA Resource Browser dialog searches for and displays <code>.jar</code> files starting in the SCA-INF/lib subdirectory of the current project directory. The JAR file includes the interface class and any supporting classes. Note: If you select a JAR file outside of the current project, Oracle JDeveloper creates a copy of the JAR file in the SCA-INF/lib directory of the current project. When prompted, click OK to accept.
Java Interface	Enter the fully qualified Java class name of the previously created Enterprise JavaBeans interface. This class must exist in the selected JAR file. If a JAR file is not specified, it is assumed that the class is in the /SCA-INF/classes subdirectory of the current project directory.
WSDL URL	Note: Ensure that you have created the annotations for the Enterprise JavaBeans interface before generating the WSDL file, as described in Section 52.2.5, "How to Use Web Service Annotations." Click the second icon to the right to generate a WSDL file that represents the Enterprise JavaBeans interface. If you created SDO objects through Oracle JDeveloper, as described in Section 52.2.1, "How to Create SDO Objects Using the SDO Compiler," ensure that you select the WSDL file that was automatically generated with this option.
Port Type	Select the port type.
Callback Port Type	Select the callback port type (for asynchronous services).

3. Click **OK**.

52.4 Designing an Enterprise JavaBeans Client to Invoke Oracle SOA Suite

To invoke an SDO - Enterprise JavaBeans service from Enterprise JavaBeans, you must use the client library. Follow these guidelines to design an Enterprise JavaBeans client.

- Look up the `SOAServiceInvokerBean` from the JNDI tree.
- Get an instance of `SOAServiceFactory` and ask the factory to return a proxy for the Enterprise JavaBeans service interface.
- You can include a client side Enterprise JavaBeans invocation library (`fabric-ejbClient.jar` or the `fabric-runtime.jar` file located in the Oracle JDeveloper home directory or Oracle WebLogic Server) in the Enterprise JavaBeans client application. For example, the `fabric-runtime.jar` file can be located in the `JDev_Home\jdeveloper\soa\modules\oracle.soa.fabric_11.1.1` directory.

If the Enterprise JavaBeans application is running in a different JVM than Oracle SOA Suite, the Enterprise JavaBeans application must reference the `ejbClient` library.

[Example 52–4](#) provides an example.

Example 52–4 Enterprise JavaBeans Client Code

```

Properties props = new Properties();
    props.put(Context.INITIAL_CONTEXT_FACTORY,
        "weblogic.jndi.WLInitialContextFactory");
    props.put(Context.PROVIDER_URL, "t3://" + HOSTNAME + ":" + PORT);
    InitialContext ctx = new InitialContext(props);
    SOAServiceInvokerBean invoker =
        (SOAServiceInvokerBean)

ctx.lookup("SOAServiceInvokerBean#oracle.integration.platform.blocks.sdox.ejb.api.
SOAServiceInvokerBean");

    //-- Create a SOAServiceFactory instance
    SOAServiceFactory serviceFactory = SOAServiceFactory.newInstance(invoker);

    //-- Get a dynamic proxy that is essentially a remote reference
    HelloInterface.ejbRemote =
serviceFactory.createService("MyTestEJBService", HelloInterface.class);

    //-- Invoke methods
    Item item = (Item) DataFactory.INSTANCE.create(Item.class);
    item.setNumber(new BigInteger("32"));
    SayHello sayHello = (SayHello)
DataFactory.INSTANCE.create(SayHello.class);
    sayHello.setItem(item);

    SayHelloResponse response =.ejbRemote.sayHello(sayHello);
    Item reply = response.getResult();

```

52.5 Specifying Enterprise JavaBeans Roles

To specify role names required to invoke SOA composite applications from any Java EE application, you add the roles names in the Enterprise JavaBeans adapter service configuration. The Enterprise JavaBeans adapter service checks to see if the caller principal has the security role.

```

<service name="EJBService" ui:wSDLLocation="BPELEJBProcess.wsdl">
    <interface.wSDL
interface="http://xmlns.oracle.com/EJBApplication/EJBProject/BPELEJBProcess#wsdl.i
nt
erface(BPELProcess1)"callbackInterface="http://xmlns.oracle.com/EJBApplication/
EJBProject/BPELEJBProcess#
wsdl.interface(BPELEJBProcessCallback)"/>
<property name="rolesAllowed">Superuser, Admin</property>
    <binding.ejb javaInterface="java.class.ejb.com" serviceId="EJBService"
        jarLocation="soaebj.jar"/>
</service>

```

52.6 Configuring JNDI Access

This section describes two methods for configuring JNDI access.

52.6.1 How to Create a Foreign JNDI

Follow these guidelines to configure JNDI access.

- You can configure a foreign JNDI provider to link a foreign JNDI tree to your local server instance and access the object as if it is local. See *Oracle Fusion Middleware Programming JNDI for Oracle WebLogic Server*.
- You can also provide JNDI environment variables as the properties for the Enterprise JavaBeans adapter reference, as shown in [Example 52–5](#). An Enterprise JavaBeans binding component enables you to create your own map or use the default EJBBC binding component map. Note that the `map` property is optional if you use EJBBC. For security reasons, the JNDI security credentials must be stored in a CSF store and be referenced as shown in [Example 52–5](#).

Example 52–5 Environment Variables for Enterprise JavaBeans Adapter Reference

```
<property name=
"java.naming.factory.initial">weblogic.jndi.WLInitialContextFactory</property>
<property name="java.naming.provider.url">t3://host:7001</property>
<property name="oracle.jps.credstore.map">default</property>
<property name="oracle.jps.credstore.key">weblogic</property>
```

The security credential can also be stored in the credential store framework. For more information, see *Oracle Fusion Middleware Security Guide*.

52.6.2 How to Create a Custom CSF Map for JNDI Lookup

If you create your own credential store framework (CSF) map instead of using the default Enterprise JavaBeans BC CSF map, you must modify the *Domain_Home/config/fmwconfig/system-jazn.data.xml* file and add the following permission to the entry for the *fabric-runtime.jar* permission grant.

```
<class>oracle.security.jps.service.credstore.CredentialAccessPermission</class>
  <name>>context=SYSTEM,mapName=*,keyName=*</name>
  <actions>*</actions>
</permission>
```

You must then restart Oracle WebLogic Server.

For more information on CSF, see *Oracle Fusion Middleware Security Guide*.

Processing Large Documents

This chapter describes best practices and limitations for processing large documents in Oracle SOA Suite.

This chapter includes the following sections:

- [Section 53.1, "Introduction to Processing Large Documents"](#)
- [Section 53.2, "Best Practices for Handling Large Documents"](#)
- [Section 53.3, "Limitations on Concurrent Processing of Large Documents"](#)

53.1 Introduction to Processing Large Documents

This document provides the best practices for processing large XML document payloads in Oracle SOA Suite service engines. Limitations on using large payloads are also described.

Oracle recommends that you follow these best practices before developing and executing large payloads.

53.2 Best Practices for Handling Large Documents

This section describes the following scenarios of handling large documents and the best practice approach for each scenario:

- [Section 53.2.1, "Setting Audit Levels from Oracle Enterprise Manager for Large Payload Processing"](#)
- [Section 53.2.2, "Using the Assign Activity in BPEL/Mediator"](#)
- [Section 53.2.3, "Using XSLT Transformations for Repeating Structures"](#)
- [Section 53.2.4, "Using Adapter Support for Streaming Large Payloads"](#)
- [Section 53.2.5, "Using Correct Settings for Large Payload Scenarios"](#)
- [Section 53.2.6, "Processing Large Documents in Oracle B2B"](#)
- [Section 53.2.7, "Setting the Default JTA Timeout in for Large Documents"](#)
- [Section 53.2.8, "Using Large Number of Activities in BPEL Processes \(Without FlowN\)"](#)
- [Section 53.2.9, "Using Large Number of Activities in BPEL Processes \(With FlowN\)"](#)
- [Section 53.2.10, "Using a Flow With Multiple Sequences"](#)
- [Section 53.2.11, "Using a Flow with One Sequence"](#)

- [Section 53.2.12, "Using Flow with No Sequence"](#)
- [Section 53.2.13, "Large Numbers of Mediators in Composites"](#)
- [Section 53.2.14, "Using XSLT Transformations on Large Payloads \(For BPEL and Mediator\)"](#)

53.2.1 Setting Audit Levels from Oracle Enterprise Manager for Large Payload Processing

For large payload processing, turn off audit level logging for the specific composite. You can set the settings/composite audit level option to **Off** or **Production** in Oracle Enterprise Manager Fusion Middleware Control Console. If you set the settings/composite audit level option to **Development**, then it serializes the entire large payload into an in-memory string and can lead to an out-of-memory error.

For more information about setting audit levels, see *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite*.

53.2.2 Using the Assign Activity in BPEL/Mediator

When using the assign activity, in BPEL or Mediator, to manipulate large payloads, do not assign the complete message. Instead, assign only the part of the payload that you need.

In addition, when using the assign activity in BPEL, Oracle recommends using local variables instead of process variables wherever possible. Local variables are limited to the scope of the BPEL process. These get deleted from memory and from the database after you close the scope. On the other hand, the life cycle of a global variable is tied with the instance life cycle. These variables stay in memory or disk until the instance completes. Thus, local variables are preferred to process or global variables.

53.2.3 Using XSLT Transformations for Repeating Structures

In scenarios where the repeating structure is of smaller payloads compared to the overall payload size, Oracle recommends using XSLT transformation because the current XSLT implementation materializes the entire DOM in memory. For example, `PurchaseOrder.LineItem.Supplier` (a subpart of a large payload).

You can also substitute it with the assign activity, as it performs a shadow copy. Although a shadow copy does not materialize DOM, it creates a shadow node to point to the source document.

You can also use the following optimized translation functions while performing transforms/translations of large payloads:

- `ora:doTranslateFromNative`
- `ora:doTranslateToNative`
- `ora:doStreamingTranslate`

For more information about the usage of these functions, see *Oracle Fusion Middleware User's Guide for Technology Adapters*.

53.2.4 Using Adapter Support for Streaming Large Payloads

The streaming feature exposed by the File/FTP adapters is used for processing large payloads. Only the Files/FTP adapters support large payload processing for both

inbound and outbound processing. The other adapters do not have explicit support for both.

Note: Only the Files/FTP adapters and web services binding support streaming. You should consider alternative strategies for handling large documents with other binding.

For more information about how the streaming feature is used for large payloads, see Section 4.5.4, "Oracle File Adapter Scalable DOM" in the *Oracle Fusion Middleware User's Guide for Technology Adapters*.

53.2.5 Using Correct Settings for Large Payload Scenarios

Add `-Dweblogic.resourcepool.max_test_wait_secs=30` in `setDomainEnv.sh` for `JAVA_OPTIONS`, and then restart the server. Without this setting, the large payload scenarios fails with `ResourceDisabledException` for the dehydration data source.

53.2.6 Processing Large Documents in Oracle B2B

For processing large documents in Oracle B2B, various parameters such as `MDSInstance Cache Size`, `Protocol Message Size`, `Number of threads`, `StuckThread Max Time`, and `Tablespace` must be tuned. The following sections describe the parameters you must set for processing large documents in Oracle B2B:

- [Section 53.2.6.1, "MDSInstance Cache Size"](#)
- [Section 53.2.6.2, "Protocol Message Size"](#)
- [Section 53.2.6.3, "Number of Threads"](#)
- [Section 53.2.6.4, "StuckThread Max Time"](#)
- [Section 53.2.6.5, "Tablespace"](#)

53.2.6.1 MDSInstance Cache Size

To set `MDSInstance` cache size, the property and value must be added in the `$DOMAIN_HOME/config/soa-infra/configuration/b2b-config.xml` file as mentioned in the example below:

```
<property>
  <name>b2b.mdsCache</name>
  <value>200000</value>
  <comment>MDS Instance cache size </comment>
</property>
```

53.2.6.2 Protocol Message Size

If Oracle B2B wants to send or receive more than 10 MB of message or import/export configuration is more than 10 MB, then the following settings must be changed accordingly in the Oracle WebLogic Server Administration Console:

```
Environment > servers > soa_server > protocols > General >
change Maximum Message Size
```

This setting can be added/modified in the `$DOMAIN_HOME/config/config.xml` file next to the server name configuration, as shown below:

```
<name>soa_server1</name>
```

```
<max-message-size>150000000</max-message-size>
```

Note: By default, `max-message-size` is not available in this `config.xml` file.

53.2.6.3 Number of Threads

This parameter helps to improve the message processing capability of Oracle B2B and must be set in the `$_DOMAIN_HOME/config/soa-infra/configuration/b2b-config.xml` file.

```
<property>
  <name>b2b.inboundProcess.threadCount</name>
  <value>5</value>
  <comment></comment>
</property>
<property>
  <name>b2b.inboundProcess.sleepTime</name>
  <value>10</value>
  <comment></comment>
</property>
<property>
  <name>b2b.outboundProcess.threadCount</name>
  <value>5</value>
  <comment></comment>
</property>
<property>
  <name>b2b.outboundProcess.sleepTime</name>
  <value>10</value>
  <comment></comment>
</property>
<property>
  <name>b2b.defaultProcess.threadCount</name>
  <value>5</value>
  <comment></comment>
</property>
<property>
  <name>b2b.defaultProcess.sleepTime</name>
  <value>10</value>
  <comment></comment>
</property>
```

53.2.6.4 StuckThread Max Time

The `StuckThread Max Time` parameter checks the number of seconds that a thread must be continually working before this server considers the thread stuck. You must change the following settings in the Oracle WebLogic Server Administration Console:

```
Environment > servers > soa_server > Configuration > Tuning >
change Stuck Thread Max Time
```

53.2.6.5 Tablespace

If you must store more than a 150 MB configuration in the data file, then you must extend or add the data file to increase the tablespace size, as follows:

```
ALTER TABLESPACE sh_mds add DATAFILE 'sh_mds01.DBF' SIZE 100M autoextend on next
  10M maxsize unlimited;
ALTER TABLESPACE sh_ias_temp add TEMPFILE 'sh_ias_temp01.DBF' SIZE 100M autoextend
```

```
on next 10M maxsize unlimited;
```

53.2.7 Setting the Default JTA Timeout in for Large Documents

Oracle recommends that the default JTA Timeout parameter be increased from the current 30 to an appropriate value for processing large documents.

53.2.8 Using Large Number of Activities in BPEL Processes (Without FlowN)

To deploy BPEL processes that have a large number of activities, for example, 50000, the following settings are required:

```
Set MEM_ARGS: -Xms512m -Xmx1024m -XX:PermSize = 128m
-XX:MaxPermSize = 256m
```

```
Number of Concurrent Threads = 20
```

```
Number of Loops = 5 Delay = 100 ms
```

The above settings enable you to deploy and execute BPEL processes, which use only while loops without the flowN activities, successfully.

53.2.9 Using Large Number of Activities in BPEL Processes (With FlowN)

To deploy BPEL processes that have large number of activities, for example, 50000, the following settings are required:

```
Set USER_MEM_ARGS: -Xms2048m -Xmx2048m -XX:PermSize=128m
-XX:MaxPermSize=256m
```

```
Number of Concurrent Threads= 10
```

```
Number of Loops=5 Delay=100 ms
```

Set `<statsLastN>-1</statsLastN>` in the `bpel-config.xml` file.

The above settings enable you to deploy and execute BPEL processes, which use the flowN activities, successfully.

For more information, see [Chapter 10, "Using Parallel Flow in a BPEL Process."](#)

53.2.10 Using a Flow With Multiple Sequences

BPEL processes, which have large numbers of activities up to 7000, can be deployed and executed successfully with the following settings:

```
Set USER_MEM_ARGS: -Xms2048m -Xmx2048m -XX:PermSize=128m
-XX:MaxPermSize=256m
```

Note: If you deploy BPEL processes with more than 8000 activities, then BPEL compilation throws errors.

53.2.11 Using a Flow with One Sequence

BPEL processes, which have large number of activities up to 7000, can be deployed and executed successfully with the following settings:

```
Set USER_MEM_ARGS: -Xms2048m -Xmx2048m -XX:PermSize=128m
-XX:MaxPermSize=512m
```

Note: If you deploy BPEL processes with more than 10,000 activities, then the BPEL compilation fails.

53.2.12 Using Flow with No Sequence

You can deploy and execute BPEL processes, which have large number of activities, for example, up to 5000, successfully.

There is a probability that the BPEL compilation could fail for 6000 activities.

53.2.13 Large Numbers of Mediators in Composites

Oracle recommends that you not have more than 50 mediators in a single composite. The JTA Transaction timeout should be increased to an appropriate high value based on the environment.

53.2.14 Using XSLT Transformations on Large Payloads (For BPEL and Mediator)

Oracle recommends that you not apply the XSLT Transformation on large payloads as this would result in out-of-memory errors when XSLT must traverse the entire document.

53.3 Limitations on Concurrent Processing of Large Documents

This section describes the limitations on concurrent processing of large documents. This section includes the following topics:

- [Section 53.3.1, "Opaque Schema for Processing Large Payloads"](#)
- [Section 53.3.2, "Streaming MTOM Attachments"](#)
- [Section 53.3.3, "Importing Large Data Sets in Oracle B2B"](#)

53.3.1 Opaque Schema for Processing Large Payloads

There is a limitation when you use an opaque schema for processing large payloads. The entire data for the opaque translator is converted to a single Base64-encoded string. An opaque schema is generally used for smaller data. For large data, use the attachments feature instead of the opaque translator.

For more information about the usage of these functions, see *Oracle Fusion Middleware User's Guide for Technology Adapters*.

53.3.2 Streaming MTOM Attachments

The incoming requests for streaming MTOM attachments that are passed through the Service Infrastructure are normalized, and the processing of such messages are not optimized inside the Service Infrastructure layer.

53.3.3 Importing Large Data Sets in Oracle B2B

Oracle recommends that you not use browsers for large data set imports and to use the command-line utility. The following utility commands are recommended for large data configuration:

- `purge` - This command is used to purge the entire repository.

- `import` - This command is used to import the specified ZIP file.
- `deploy` - This command is used to deploy an agreement with whatever name is specified. If no name is specified, then all the agreements are deployed.

However, the `purgeimportdeploy` option is not recommended to be used for transferring or deploying Oracle B2B configuration.

Part IX

Appendices

This part describes Oracle SOA Suite appendixes.

This part contains the following appendixes:

- [Appendix A, "BPEL Process Activities and Services"](#)
- [Appendix B, "XPath Extension Functions"](#)
- [Appendix C, "Deployment Descriptor Properties"](#)
- [Appendix D, "Understanding Sensor Public Views and the Sensor Actions XSD"](#)
- [Appendix E, "Oracle BAM Web Services Operations"](#)
- [Appendix F, "Oracle BAM Alert Rule Options"](#)
- [Appendix G, "Oracle BAM ICommand Operations and File Formats"](#)
- [Appendix H, "Normalized Message Properties"](#)
- [Appendix I, "Oracle User Messaging Service Applications"](#)

BPEL Process Activities and Services

This appendix describes the activities and services that you use when designing a BPEL process in a SOA composite application.

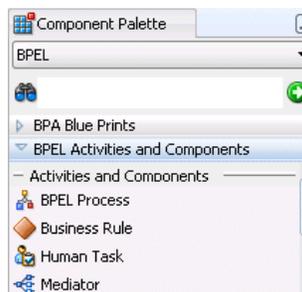
This appendix includes the following sections:

- [Section A.1, "Introduction to Activities and Components"](#)
- [Section A.2, "Introduction to BPEL Activities"](#)
- [Section A.3, "Introduction to BPEL Services"](#)
- [Section A.4, "Publishing and Browsing the Oracle Service Registry"](#)
- [Section A.5, "Validating When Loading a Process Diagram"](#)

A.1 Introduction to Activities and Components

When you expand **BPEL Activities and Components** in the Component Palette of Oracle BPEL Designer, service components display under the **Activities and Components** header.

Figure A-1 Activities and Components



See the following sections for additional details.

- BPEL process
[See Part II, "Using the BPEL Process Service Component"](#)
- Business rule
[See Part IV, "Using the Business Rules Service Component"](#)
- Human task
[Section 25.4, "Associating the Human Task Service Component with a BPEL Process."](#)

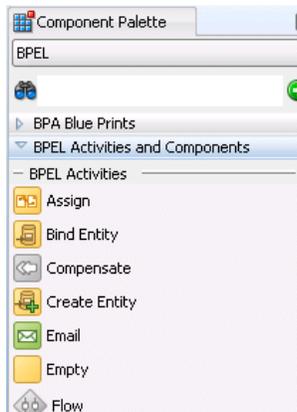
- Mediator
See [Part III, "Using the Oracle Mediator Service Component"](#)

A.2 Introduction to BPEL Activities

Oracle BPEL Designer includes activities that are available for dragging into a BPEL process. These activities enable you to perform specific tasks within a process. This section provides a brief overview of these activities and provides references to other documentation that describes how to use these activities.

To access these activities, expand **BPEL Activities and Components** in the Component Palette of Oracle BPEL Designer. The activities display under the **BPEL Activities** header.

Figure A–2 BPEL Activities



For more information about activities, see the *Business Process Execution Language for Web Services Specification* by visiting the following URL:

<http://www.oasis-open.org>

A.2.1 Tabs Common to Many Activities

While each activity performs specific tasks, many activities include tabs that enable you to perform similar tasks. This section describes these common tabs.

- The **Sensors** tab displays on all activities and enables you to create sensors for capturing details about an activity.
- The **Correlations** tab displays in invoke, receive, and reply activities, the onMessage branch of pick activities, and the OnMessage variant of event handlers. Correlation sets address complex interactions between a process and its partners by providing a method for explicitly specifying correlated groups of operations within a service instance. A set of correlation tokens is defined as a set of properties shared by all messages in the correlated group.
- The **Properties** tab displays in invoke, receive, and reply activities, and the onMessage branch of pick activities. You create header variables for use with the Oracle JCA adapters.
- The **Annotations** tab displays on all activities and enables you to provide descriptions in activities in the form of code comments and name and pair value assignments.

Note the following issues when using annotations in Oracle JDeveloper:

- The **Annotations** tab in activities of Oracle JDeveloper does not provide a method for changing the order of annotations. As a work around, change the order of annotations in the **Source** view of the project's BPEL file in Oracle BPEL Designer.
- The otherwise branch in a switch activity does not allow you to create annotations. However, the case branch in a switch activity does provide this functionality.

For more information about these tabs, see the following:

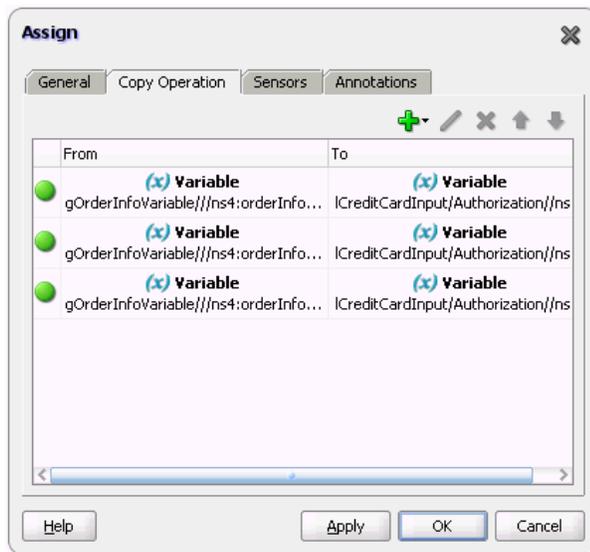
- The online help for these tabs for additional details about their use
- [Section 9.4, "Using Correlation Sets in an Asynchronous Service"](#)
- [Chapter 17, "Using Oracle BPEL Process Manager Sensors"](#)
- [Appendix H, "Normalized Message Properties"](#)
- *Oracle Fusion Middleware User's Guide for Technology Adapters*

A.2.2 Assign Activity

This activity provides a method for data manipulation, such as copying the contents of one variable to another. This activity can contain any number of elementary assignments.

[Figure A-3](#) shows the Assign dialog. You can perform the following tasks:

- Click the **General** tab to provide the assign activity with a meaningful name.
- Click the **Copy Operation** tab and the **Add** icon (shown in [Figure A-3](#)), and then select **Copy Operation** from the dropdown list to access the Create Copy Operation dialog. This action enables you to copy the contents of the source element (variable, expression, XML fragment, or partner link) in the **From** field to the contents of the destination element in the **To** field. You can also select a part (typically the payload) and an XPath query (a language for addressing parts of an XML document). Other selections such as **Append Operation**, **Insert-After Operation**, and others are also available from this list.

Figure A-3 Copy Operations Tab of Assign Activity Dialog

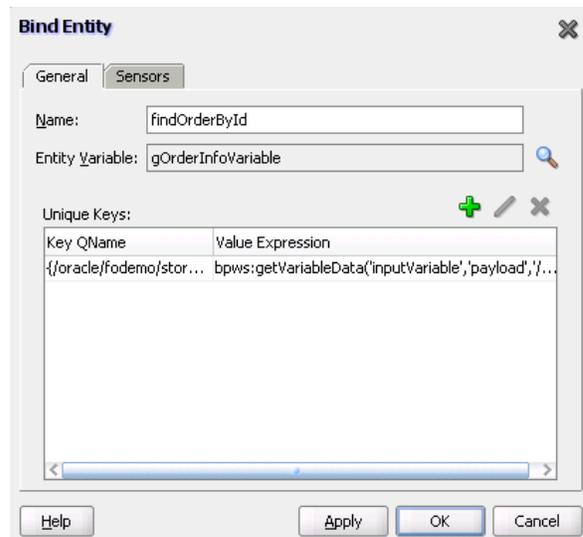
If an assign activity contains multiple `bpelx:append` settings, it must be split into two assign activities. Otherwise, the `bpelx:append` is moved to the end of the list each time, which can cause problems. As a work around, move it manually.

For more information about the assign activity, see [Chapter 7, "Manipulating XML Data in a BPEL Process."](#)

A.2.3 Bind Entity Activity

This activity enables you to select the entity variable to act as the data handle to access and plug in different data provider service technologies.

The entity variable can be used with an Oracle Application Development Framework (ADF) Business Component data provider service using service data object (SDO)-based data. The entity variable enables you to specify BPEL data operations to be performed by an underlying data provider service. The data provider service performs the data operations in a data store behind the scenes and without use of other data store-related features provided by Oracle BPEL Process Manager (for example, the database adapter). This action enhances Oracle BPEL Process Manager runtime performance and incorporates native features of the underlying data provider service during compilation and runtime.

Figure A-4 Bind Entity Dialog

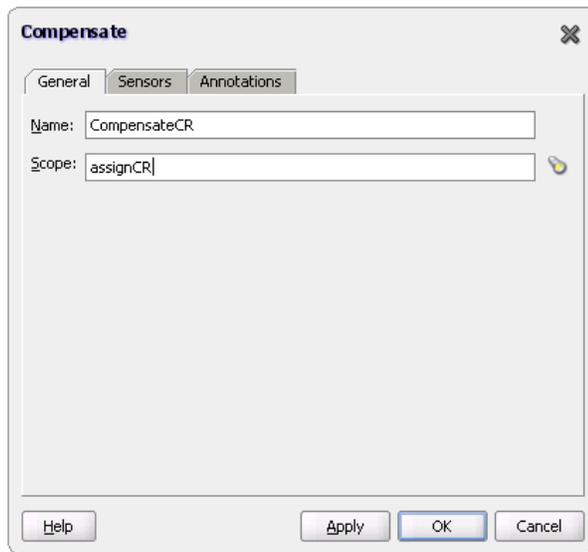
A.2.4 Compensate Activity

This activity invokes compensation on an inner scope activity that has successfully completed. This activity can be invoked only from within a fault handler or another compensation handler. Compensation occurs when a process cannot complete several operations after completing others. The process must return and undo the previously completed operations. For example, assume a process is designed to book a rental car, a hotel, and a flight. The process books the car and the hotel, but cannot book a flight for the correct day. In this case, the process performs compensation by unbooking the car and the hotel.

The compensation handler is invoked with the compensate activity, which names the scope on which the compensation handler is to be invoked.

Figure A-5 shows the Compensate dialog. You can perform the following tasks:

- Click the **General** tab to provide the activity with a meaningful name.
- Select the **scope** activity on which the compensation handler is to be invoked.

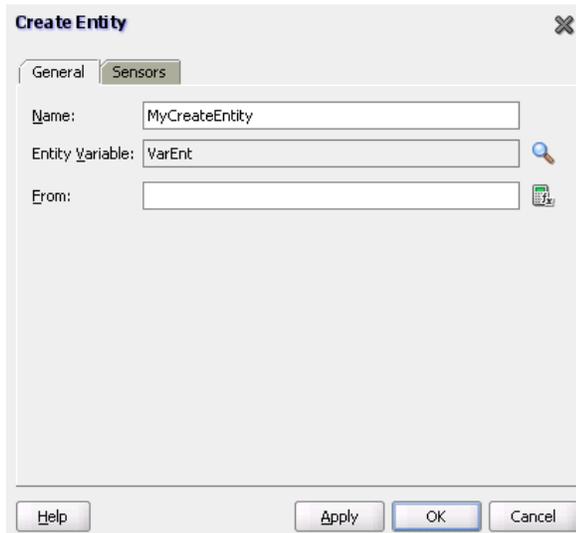
Figure A–5 Compensate Dialog

The screenshot shows a dialog box titled "Compensate". It has three tabs: "General", "Sensors", and "Annotations". The "General" tab is selected. Inside the dialog, there are two text input fields. The first is labeled "Name:" and contains the text "CompensateCR". The second is labeled "Scope:" and contains the text "assignCR". At the bottom of the dialog, there are four buttons: "Help", "Apply", "OK", and "Cancel".

For more information about the compensate activity, see [Section 12.10, "Using Compensation After Undoing a Series of Operations."](#)

A.2.5 Create Entity

This activity enables you to create an entity variable. The entity variable can be used with an Oracle ADF Business Component data provider service using service data object (SDO)-based data.

Figure A–6 Create Entity Dialog

The screenshot shows a dialog box titled "Create Entity". It has two tabs: "General" and "Sensors". The "General" tab is selected. Inside the dialog, there are three text input fields. The first is labeled "Name:" and contains the text "MyCreateEntity". The second is labeled "Entity Variable:" and contains the text "VarEnt". The third is labeled "From:" and is empty. At the bottom of the dialog, there are four buttons: "Help", "Apply", "OK", and "Cancel".

For more information, see [Section 7.2, "Delegating XML Data Operations to Data Provider Services."](#)

A.2.6 Email Activity

This activity enables you to send an email notification about an event.

For example, an online shopping business process of an online bookstore sends a courtesy email message to you after the items are shipped. The business process calls the notification service with your user ID and notification message. The notification service gets the email address from Oracle Internet Directory.

Figure A-7 shows the Email dialog.

Figure A-7 Email Dialog

The screenshot shows the 'Email' dialog box with the following fields and values:

- From Account:** Default
- To:** eriaResponse/ns6:result/ns4:ConfirmedEmail)%>
- Cc:** (empty)
- Bcc:** (empty)
- Reply To:** (empty)
- Subject:** iable',/ns4:orderInfoVOSDO/ns4:OrderId)%> shipped!
- Body:** s4:FirstName)%>,
 your order has been shipped.

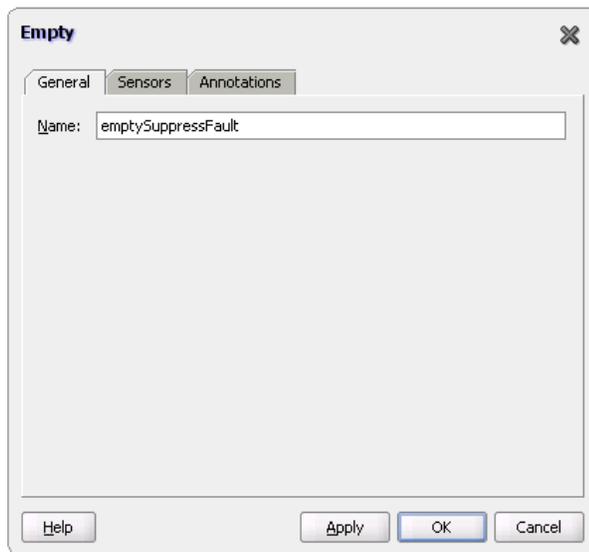
At the bottom of the dialog, there is a checkbox for 'Multipart message' and a text field for 'attachments'. The 'OK' button is highlighted.

For more information about the email activity, see [Section 16.3.1, "How To Configure the Email Notification Channel."](#)

A.2.7 Empty Activity

This activity enables you to insert a no-operation instruction into a process. This activity is useful when you must use an activity that does nothing (for example, when a fault must be caught and suppressed).

Figure A-8 shows the Empty dialog.

Figure A–8 Empty Dialog

For more information about the empty activity, see [Section 12.9.7, "How to Create an Empty Activity to Insert No-Op Instructions into a Business Process."](#)

A.2.8 Flow Activity

This activity enables you to specify one or more activities to be performed concurrently. A flow activity completes when all activities in the flow have finished processing. Completion of a flow activity includes the possibility that it can be skipped if its enabling condition is false.

For example, assume you use a flow activity to enable two loan offer providers (United Loan service and Star Loan service) to start in parallel. In this case, the flow activity contains two parallel activities – the sequence to invoke the United Loan service and the sequence to invoke the Star Loan service. Each service can take an arbitrary amount of time to complete their loan processes.

[Figure A–9](#) shows an initial flow activity with its two panels for parallel processing. You drag activities into both panels to create parallel processing. When complete, a flow activity looks like that shown in [Figure A–10](#).

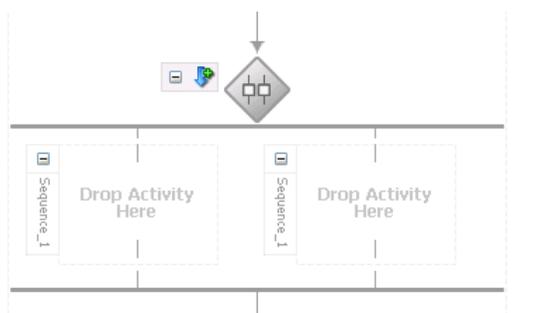
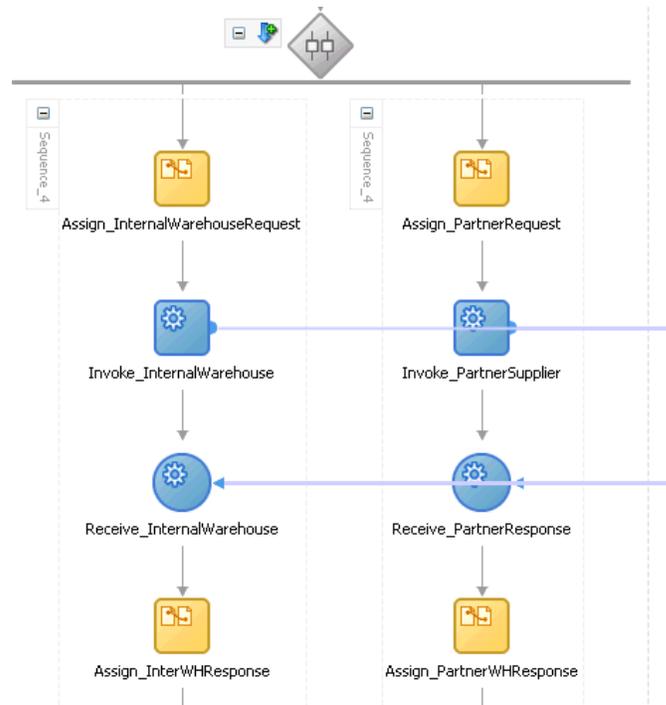
Figure A–9 Flow Dialog (At Time of Creation)

Figure A-10 Flow Dialog (After Design Completion)

Note: Oracle's BPEL implementation executes flows in the same, single execution thread of the BPEL process and not in separate threads.

For more information about the flow activity, see [Section 10.2, "Creating a Parallel Flow."](#)

A.2.9 FlowN Activity

This activity enables you to create multiple flows equal to the value of N , which is defined at runtime based on the data available and logic within the process. An index variable increments each time a new branch is created, until the index variable reaches the value of N .

[Figure A-11](#) shows the FlowN dialog.

Figure A–11 FlowN Dialog

The FlowN dialog window has three tabs: General, Sensors, and Annotations. The General tab is active. It contains the following fields:

- Name: FlowNCR2
- N: 2
- Index Variable: FlowNCR2_Variable

At the bottom of the dialog are four buttons: Help, Apply, OK, and Cancel.

For more information about the flowN activity, see [Section 10.3, "Customizing the Number of Flow Activities with the flowN Activity."](#)

A.2.10 IM Activity

This activity enables you to send an automatic, asynchronous instant message (IM) notification to a user, group, or destination address. [Figure A–12](#) shows the IM dialog.

Figure A–12 IM Dialog

The IM dialog window has two tabs: IM and Sensors. The IM tab is active. It contains the following fields:

- To: jstein
- Body: For your approval

Below the body field, there is a note: "Message body should be plain text." At the bottom of the dialog are four buttons: Help, Apply, OK, and Cancel.

For more information, see [Section 16.3.2, "How to Configure the IM Notification Channel."](#)

A.2.11 Invoke Activity

This activity enables you to specify an operation you want to invoke for the service (identified by its partner link). The operation can be one-way or request-response on a port provided by the service. You can also automatically create variables in an invoke activity. An invoke activity invokes a synchronous web service or initiates an asynchronous web service.

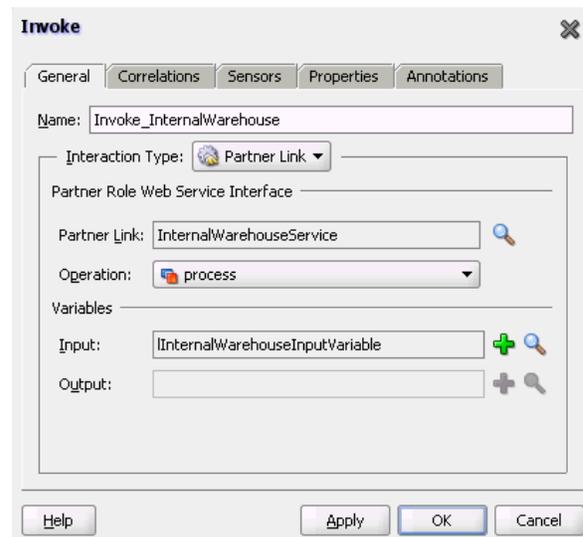
The invoke activity opens a port in the process to send and receive data. It uses this port to submit required data and receive a response. For synchronous callbacks, only one port is needed for both the send and the receive functions.

The invoke activity supports the `bpelx:inputProperty` and `bpelx:outputProperty` that facilitate the passing of properties through the SOAP header and the obtaining of SOA runtime system properties for useful information such as the `tracking.compositeInstanceId` and `tracking.conversationId`.

Figure A–13 shows the Invoke dialog. You can perform the following tasks:

- Provide the activity with a meaningful name.
- Select the partner link for which to specify an operation.
- Select the operation to be performed.
- Automatically create a variable or select an existing variable in which to transport the data (payload).

Figure A–13 Invoke Dialog



For more information about the invoke activity, see the following:

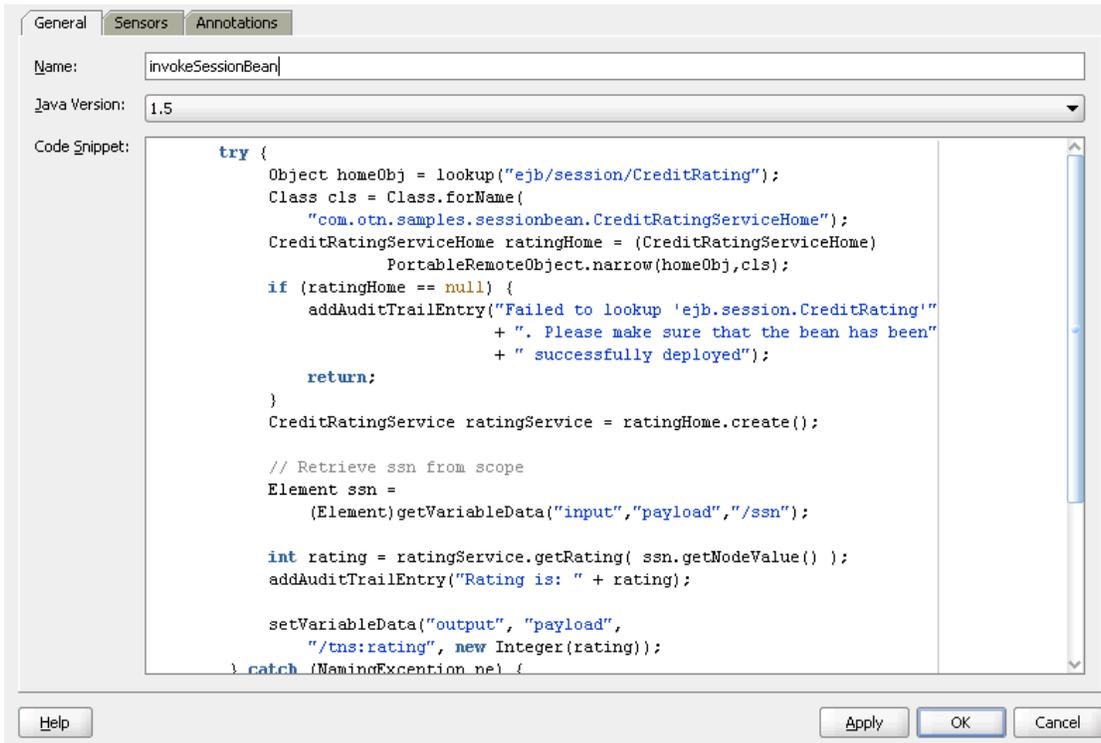
- [Section 8.2.2.3, "Invoke Activity for Performing a Request"](#)
- [Section 9.2.1.2, "Adding an Invoke Activity"](#)
- [Section 12.8.2, "How to Return a Fault in an Asynchronous Interaction"](#)
- [Chapter 6, "Introduction to Interaction Patterns in a BPEL Process"](#)

A.2.12 Java Embedding Activity

This activity enables you to add custom Java code to a BPEL process using the Java BPEL `exec` extension `<bpelx:exec>`. This is useful when you have Java code that can perform a function, and want to use this existing code instead of starting over.

Figure A–14 shows the Edit Java Embedding dialog.

Figure A–14 Edit Java Embedding Dialog



For more information about the Java embedding activity, see [Chapter 13, "Incorporating Java and Java EE Code in a BPEL Process."](#)

A.2.13 Phase Activity

This activity creates Oracle Mediator and business rules service components for integration with a BPEL process. You create message request input and message response output variables and design business rules for evaluating variable content for the BPEL process.

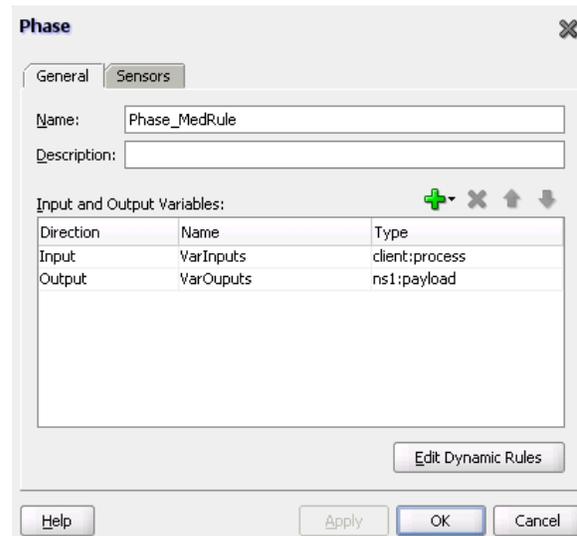
When you complete these tasks, the following activities and service components are created:

- An assign activity that includes the message request input and message response output variables.
- An invoke activity, which is automatically designed to invoke an Oracle Mediator partner link in the BPEL process.
- The Oracle Mediator partner link, which is automatically designed to route the message request input variable to the business rules service component in the SOA composite application of which this BPEL process is a part. The business rules service component displays in the SOA Composite Editor. Oracle Mediator also displays as a service component in the SOA Composite Editor.

- The business rules service component, which evaluates the content of the message request input variable and returns the results in the message response output variable to Oracle Mediator. Oracle Mediator then makes a routing decision and routes the message to the correct target destinations.

Figure A–15 shows Phase dialog.

Figure A–15 Phase Dialog



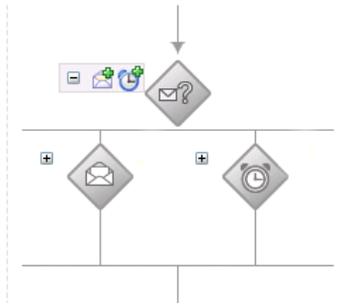
A.2.14 Pick Activity

This activity waits for the occurrence of one event in a set of events and performs the activity associated with that event. The occurrence of the events is often mutually exclusive (the process either receives an acceptance or rejection message, but not both). If multiple events occur, the selection of the activity to perform depends on which event occurred first. If the events occur nearly simultaneously, there is a race and the choice of activity to be performed is dependent on both timing and implementation.

The pick activity provides two branches, each one with a condition. When you double-click the **Pick** icon, the activity shown in Figure A–16 appears and displays two branches:

- **onMessage** (on the left)
 - Contains the code for receiving a reply, for example, from a loan service.
- **onAlarm** (on the right)
 - Contains the code for a time out, for example, after one minute.

Whichever branch completes first is executed; the other branch is not executed. The branch that has its condition satisfied first is executed.

Figure A–16 Pick Dialog

If you add correlations to an **OnMessage** branch, the correlations syntax is placed *after* the **assign** activity syntax. The correlation syntax must go *before* the **assign** activity.

As a work around, perform the following steps:

1. Create a correlation set in Oracle JDeveloper.
2. Assign this to the **OnMessage** branch.
3. Complete the remaining design tasks.
4. Before making or deploying the BPEL process, move the correlation syntax before the **assign** activity in the BPEL source code.

For more information about the pick activity, see the following:

- [Section 14.2, "Creating a Pick Activity to Select Between Continuing a Process or Waiting"](#)
- [Section 14.4, "Setting Timeouts for Synchronous Processes"](#)
- [Chapter 6, "Introduction to Interaction Patterns in a BPEL Process"](#)

A.2.15 Receive Activity

This activity specifies the partner link from which to receive information and the port type and operation for the partner link to invoke. This activity waits for an asynchronous callback response message from a service, such as a loan application approver service. While the BPEL process is waiting, it is dehydrated (compressed and stored) until the callback message arrives. The contents of this response are stored in a response variable in the process.

The receive activity supports the `bpelx:property` extensions that facilitate the passing of properties through the SOAP header, and the obtaining of SOA runtime system properties for useful information such as `tracking.compositeInstanceId` and `tracking.conversationId`.

[Figure A–17](#) shows the Receive dialog. You can perform the following tasks:

- Provide a meaningful name.
- Select the partner link service for which to specify an operation.
- Select the operation to be performed.
- Automatically create a variable or select an existing variable in which to transport the callback response.

Figure A-17 Receive Dialog

Receive

General Correlations Sensors Properties Annotations

Name: Receive_InternalWarehouse

Interaction Type: Partner Link

My Role Web Service Interface

Partner Link: InternalWarehouseService

Operation: processResponse

Variable: InternalWarehouseResponseVariable

Create Instance

Help Apply OK Cancel

For more information about the receive activity, see the following:

- ["Adding a Receive Activity"](#)
- [Chapter 6, "Introduction to Interaction Patterns in a BPEL Process"](#)

A.2.16 Receive Signal Activity

Use this activity in detail processes to wait for the notification signal from the master process to begin processing and use in a master process to wait for the notification signal from all detail processes indicating that processing has completed.

Figure A-18 Receive Signal Dialog

Receive Signal

General Sensors

Name: waitForDetailProcess

Label: completeDetailProcess

To: details

Help Apply OK Cancel

For more information, see [Chapter 15, "Coordinating Master and Detail Processes."](#)

A.2.17 Remove Entity Activity

This activity enables you to remove an entity variable. This action removes the row. [Figure A-19](#) shows the Remove Entity dialog.

Figure A-19 Remove Entity

A.2.18 Reply Activity

This activity allows the process to send a message in reply to a message that was received through a receive activity. The combination of a receive activity and a reply activity forms a request-response operation on the WSDL port type for the process.

[Figure A-20](#) shows the Reply dialog.

Figure A-20 Reply Dialog

For more information about the reply activity, see the following:

- [Section 12.8.1, "How to Return a Fault in a Synchronous Interaction"](#)
- [Chapter 6, "Introduction to Interaction Patterns in a BPEL Process"](#)

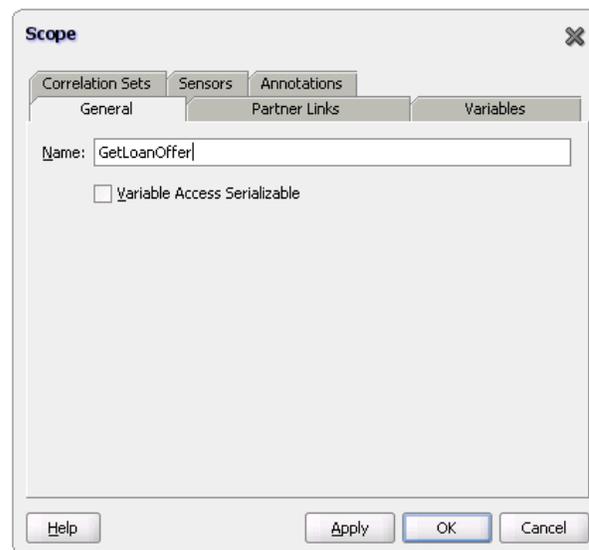
A.2.19 Scope Activity

This activity consists of a collection of nested activities that can have their own local variables, fault handlers, compensation handlers, and so on. A scope activity is analogous to a { } block in a programming language.

Each scope has a primary activity that defines its behavior. The primary activity can be a complex structured activity, with many nested activities within it to arbitrary depth. The scope is shared by all the nested activities.

[Figure A-21](#) shows the Scope dialog. Define appropriate activities inside the scope activity.

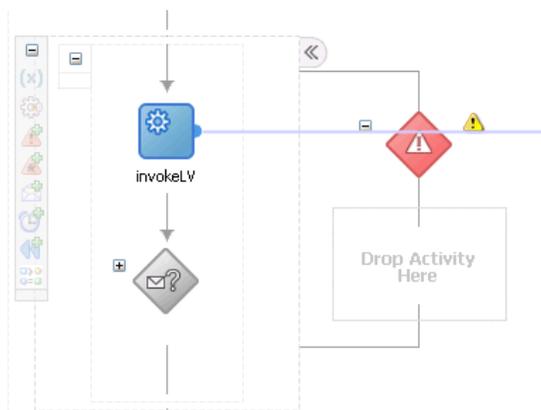
Figure A-21 Scope Dialog



Fault handling is associated with a scope activity. The goal is to undo the incomplete and unsuccessful work of a scope activity in which a fault has occurred. You define catch activities in a scope activity to create a set of custom fault-handling activities. Each catch activity is defined to intercept a specific type of fault.

[Figure A-22](#) shows the **Add Catch Branch** icon inside a scope activity. [Figure A-23](#) shows the catch activity area that appears when you click the **Add Catch Branch** icon. Within the area defined as **Drop Activity Here**, you drag additional activities to create fault handling logic to catch and manage exceptions.

For example, a client provides a social security number to a Credit Rating service when applying for a loan. This number is used to perform a credit check. If a bad credit history is identified or the social security number is identified as invalid, an assign activity inside the catch activity notifies the client of the loan offer rejection. The entire loan application process is terminated with a terminate activity.

Figure A–22 Creating a Catch Branch**Figure A–23 Catch Activity Icon**

For more information about the scope activity and fault handling, see the following:

- [Section 12.9, "Using a Scope Activity to Manage a Group of Activities"](#)
- [Chapter 6, "Introduction to Interaction Patterns in a BPEL Process"](#)

A.2.20 Sequence Activity

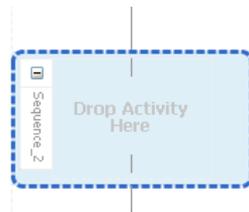
This activity enables you to define a collection of activities to be performed in sequential order. For example, you may want the following activities performed in a specific order:

- A customer request is received in a receive activity.
- The request is processed inside a flow activity that enables concurrent behavior.
- A reply message with the final approval status of the request is sent back to the customer in a reply activity.

A sequence activity makes the assumption that the request can be processed in a reasonable amount of time, justifying the requirement that the invoker wait for a synchronous response (because this service is offered as a request-response operation).

When this assumption cannot be made, it is better to define the customer interaction as a pair of asynchronous message exchanges.

When you double-click the **Sequence** icon, the activity area shown in [Figure A–24](#) appears. Drag and define appropriate activities inside the sequence activity.

Figure A–24 Sequence Activity

For more information about the sequence activity, see the following:

- [Section 10.2, "Creating a Parallel Flow"](#)
- [Chapter 6, "Introduction to Interaction Patterns in a BPEL Process"](#)

A.2.21 Signal Activity

This activity is used in a master process to notify detail processes to perform processing at runtime and used in detail processes to notify a master process that processing has completed. [Figure A–25](#) shows the Signal dialog.

Figure A–25 Signal Dialog

For more information, see [Chapter 15, "Coordinating Master and Detail Processes."](#)

A.2.22 SMS Activity

This activity enables you to send a short message system (SMS) notification about an event.

[Figure A–26](#) shows the SMS dialog.

Figure A–26 SMS Dialog

For more information about the SMS activity, see [Section 16.3.3, "How to Configure the SMS Notification Channel."](#)

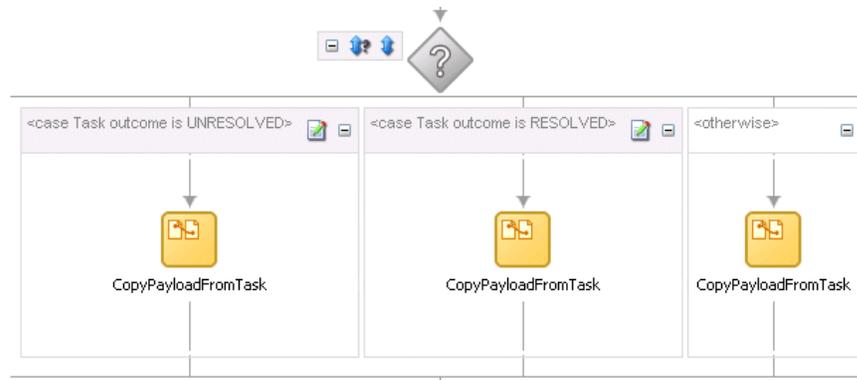
Note: The fax and pager activities are not supported in 11g Release 1 (11.1.1).

A.2.23 Switch Activity

This activity consists of an ordered list of one or more conditional branches defined in a case branch, followed optionally by an otherwise branch. The branches are considered in the order in which they appear. The first branch whose condition is true is taken and provides the activity performed for the switch. If no branch with a condition is taken, then the otherwise branch is taken. If the otherwise branch is not explicitly specified, then an otherwise branch with an empty activity is assumed to be available. The switch activity is complete when the activity of the selected branch completes.

A switch activity differs in functionality from a flow activity. For example, a flow activity enables a process to gather two loan offers at the same time, but does not compare their values. To compare and make decisions on the values of the two offers, a switch activity is used. The first branch is executed if a defined condition (inside the case branch) is met. If it is not met, the otherwise branch is executed.

[Figure A–27](#) shows a switch activity with the following defined branches.

Figure A–27 Switch Activity

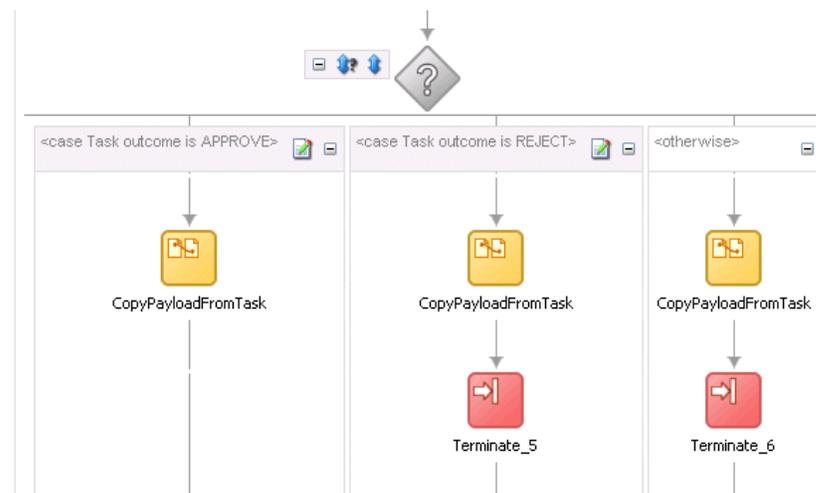
For more information about the switch activity, see the following:

- [Section 11.2, "Creating a Switch Activity to Define Conditional Branching"](#)
- [Chapter 6, "Introduction to Interaction Patterns in a BPEL Process"](#)

A.2.24 Terminate Activity

A terminate activity enables you to end the tasks of an activity (for example, the fault handling tasks in a catch branch). For example, if a client's bad credit history is identified or a social security number is identified as invalid, a loan application process is terminated, and the client's loan application document is never submitted to the service loan providers.

[Figure A–28](#) shows several terminate activities in the otherwise branch of a switch activity.

Figure A–28 Terminate Activity

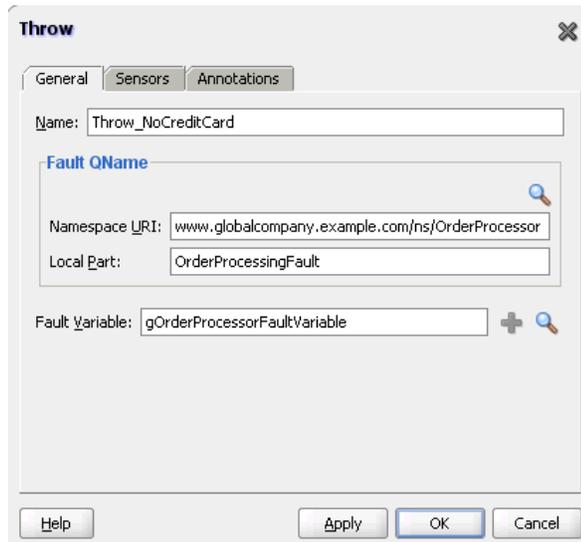
For more information about the terminate activity, see [Section 12.11, "Using the Terminate Activity to Stop a Business Process Instance."](#)

A.2.25 Throw Activity

This activity generates a fault from inside the business process.

Figure A–29 shows the Throw dialog.

Figure A–29 *Throw Dialog*



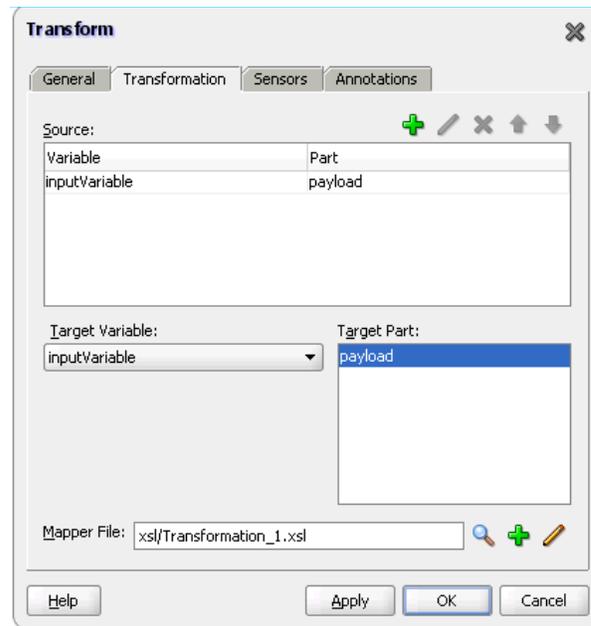
For more information about the throw activity, see [Section 12.7, "Throwing Internal Faults."](#)

A.2.26 Transform Activity

This activity enables you to create a transformation that maps source elements to target elements (for example, incoming purchase order data into outgoing purchase order acknowledgment data).

Figure A–30 shows the Transform dialog. This dialog enables you to perform the following tasks:

- Define the source and target variables and parts to map.
- Specify the transformation mapper file.
- Click the second icon (the **Add** icon) to the right of the **Mapper File** field to access the XSLT Mapper for creating a new XSL file for graphically mapping source and target elements. Click the **Edit** icon (third icon) to edit an existing XSL file.

Figure A–30 Transform Dialog

For more information about the transform activity, see [Chapter 45, "Creating Transformations with the XSLT Mapper."](#)

A.2.27 User Notification

This activity enables you to design a BPEL process in which you do not explicitly select a notification channel during design time, but simply indicate that a notification must be sent. The channel to use for sending notifications is resolved at runtime based on preferences defined by the end user in the User Messaging Preferences user interface of the Oracle User Messaging Service. This moves the responsibility of notification channel selection from Oracle BPEL Designer to the end user. If the end user does not select a preferred channel or rule, email is used by default for sending notifications to that user. [Figure A–31](#) shows the User Notification dialog.

Figure A–31 User Notification Dialog

User Notification

General Advanced Sensors

To: jstein.jcooper

Use commas to separate multiple names

Subject: Purchase Order Change Notification

Notification Message: The purchase has exceeded \$10,000

Applies to Voice, SMS, Email and IM.

Help Apply OK Cancel

A.2.28 Voice Activity

This activity enables you to send a telephone voice notification about an event.

[Figure A–32](#) shows the Voice dialog.

Figure A–32 Voice Dialog

Voice

Pager Sensors

Telephone Number: 18003687000

Body: Testing voice Message

Message body can be plain text or VXML

Help Apply OK Cancel

For more information about the voice activity, see [Section 16.3.4, "How to Configure the Voice Notification Channel."](#)

A.2.29 Wait Activity

This activity allows a process to specify a delay for a certain period or until a certain deadline is reached. A typical use of this activity is to invoke an operation at a certain

time. This activity enables you to wait for a given time period or until a certain time has passed. Exactly one of the expiration criteria must be specified.

Figure A-33 shows the Wait dialog.

Figure A-33 Wait Dialog

The image shows a 'Wait' dialog box with the following details:

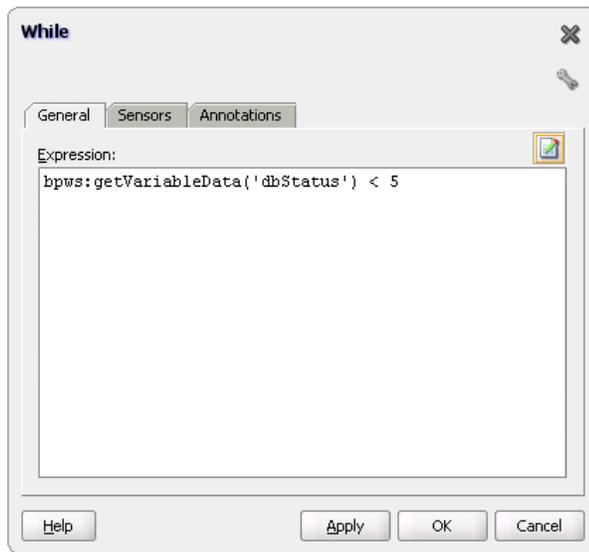
- Title:** Wait
- General Tab:**
 - Name:** Wait_30_Sec
 - For Section:**
 - Time: 0 Yrs, 0 Mons, 0 Days, 0 Hrs, 0 Mins, 31 Secs
 - Expression: [Empty text box]
 - Until Section:**
 - Time (MM/dd/yyyy HH:mm:ss): 01/10/2008 08:53:40
 - Expression: [Empty text box]
- Buttons:** Help, Apply, OK, Cancel

For more information about the wait activity, see [Section 14.3, "Creating a Wait Activity to Set an Expiration Time."](#)

A.2.30 While Activity

This activity supports repeated performance of a specified iterative activity. The iterative activity is repeated until the given `while` condition is no longer true.

Figure A-34 shows the While dialog. You can enter expressions in this dialog.

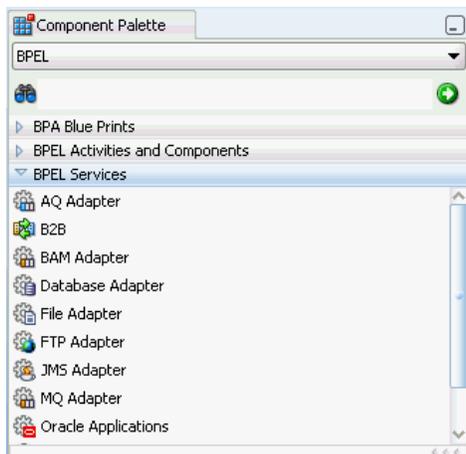
Figure A–34 While Dialog

For more information about the while activity, see [Section 11.3, "Creating a While Activity to Define Conditional Branching."](#)

A.3 Introduction to BPEL Services

BPEL processes can communicate with web-based applications and clients through web services, JCA adapters, Oracle B2B services, Oracle Business Activity Monitoring, and partner links.

To access these services, expand **BPEL Activities and Components** in the Component Palette of Oracle BPEL Designer. Then expand **BPEL Services** to display the services.

Figure A–35 BPEL Services

For more information about the adapters described in the following sections, see *Oracle Fusion Middleware User's Guide for Technology Adapters*.

A.3.1 AQ Adapter

This adapter acts as both a dequeue (inbound) and enqueue (outbound) messaging adapter. In the inbound direction, the adapter polls the queues for messages to dequeue from a destination. In the outbound direction, the adapter enqueues messages to the queue for subscribers to dequeue.

A.3.2 Oracle B2B

This adapter enables you to browse B2B metadata in the Metadata Service (MDS) repository and select document definitions.

Oracle B2B is an e-commerce gateway that enables the secure and reliable exchange of transactions between an organization and its external trading partners. Oracle B2B and Oracle SOA Suite are designed for e-commerce business processes that require process orchestration, error mitigation, and data translation and transformation within an infrastructure that addresses the issues of security, compliance, visibility, and management.

For more information, see *Oracle Fusion Middleware User's Guide for Oracle B2B*.

A.3.3 Oracle BAM Adapter

This adapter integrates Java EE applications with Oracle BAM Server to send data. This adapter is used as a reference binding component in an SOA composite application.

For more information, see *Oracle Fusion Middleware User's Guide for Oracle Business Activity Monitoring* and [Part VI, "Using Oracle Business Activity Monitoring"](#).

A.3.4 Database Adapter

This adapter enables a BPEL process to communicate with Oracle databases or third-party databases through JDBC. 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.

A.3.5 File Adapter

This adapter acts as both an inbound and outbound adapter. In the inbound direction, the adapter polls for files in a directory to retrieve and process. In the outbound direction, the adapter creates files in a directory.

A.3.6 FTP Adapter

This adapter acts as both an inbound and outbound adapter. In the inbound direction, the adapter polls for files in a directory to retrieve and process. In the outbound direction, the adapter creates files in a directory.

A.3.7 JMS Adapter

This adapter acts as both a consume (inbound) and produce (outbound) messaging adapter. In the inbound direction, the adapter polls (consumes) messages from a JMS

destination. In the outbound direction, the adapter sends (produces) messages to a JMS destination.

A.3.8 MQ Adapter

This adapter provides message exchange capabilities between BPEL processes and the IBM MQSeries messaging software.

A.3.9 Oracle Applications

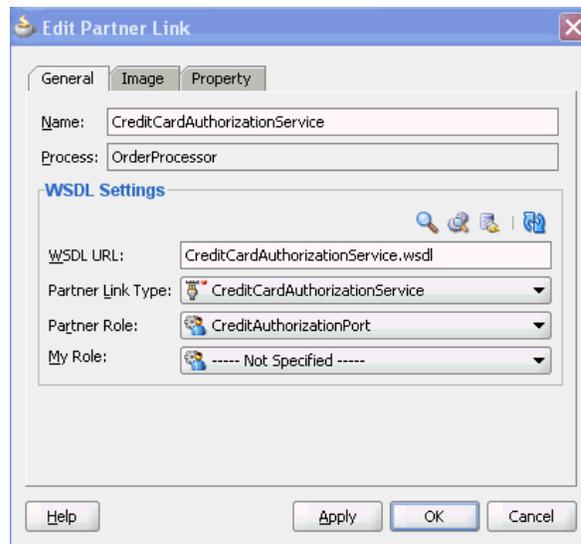
This adapter provides comprehensive, bidirectional, multimodal, synchronous, and asynchronous connectivity to Oracle Applications. The adapter supports all modules of Oracle Applications for versions 11.5.1 to 12. The adapter provides real-time and bidirectional connectivity to Oracle Applications through interface tables, views, application programming interfaces (APIs), and XML Gateway. The adapter inserts data into Oracle Applications using interface tables and APIs. To retrieve data from Oracle Applications, the adapter uses views. In addition, it uses XML Gateways for bidirectional integration with Oracle Applications. XML Gateways are also used to insert and receive Open Application Group Integration Specification (OAGIS)-compliant documents from Oracle Applications.

A.3.10 Partner Link (Web Service/Adapter)

This service enables you to define the external services with which your 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 conversation. For example, if you are creating a process to interact with a Credit Rating Service and two loan provider services (United Loan and Star Loan), you create partner links for all three services.

[Figure A-36](#) shows the Partner Link dialog. You provide the following details:

- A meaningful name for the service.
- The web services description language (WSDL) file of the external service.
- The actual service type (defined as **Partner Link Type**).
- The role of the service (defined as **Partner Role**).
- The role of the process requesting the service (defined as **My Role**).

Figure A–36 PartnerLink Activity

For more information about partner links, see the following:

- [Chapter 9, "Invoking an Asynchronous Web Service from a BPEL Process"](#)
- [Section 9.2.1.1, "Adding a Partner Link for an Asynchronous Service"](#)

A.3.11 Socket Adapter

This adapter enables you to model standard or nonstandard protocols for communication over TCP/IP sockets. You can use this adapter to create a client or server socket, and establish a connection. The data that is transported can be text or binary.

A.4 Publishing and Browsing the Oracle Service Registry

The Oracle Service Registry (OSR) provides a common standard for publishing and discovering information about web services. This section describes how to configure your Oracle SOA Suite environment to use OSR.

You can use Oracle SOA Suite with the following versions of OSR:

- OSR 10.3 (with Oracle WebLogic Server 10.3)
- OSR 10.1.3

A.4.1 How to Publish a Business Service

This section provides an overview of how to publish a business service. For specific instructions, see the documentation at the following URL:

<http://www.oracle.com/technology/tech/soa/uddi/index.html>

To publish a business service:

1. Go to the Registry Control:


```
http://hostname:port/registry/uddi/web
```
2. Click **Publish > WSDL**.

3. Log in when prompted.
4. Complete the fields on this page to specify the access point URL and publish the WSDL for the business service.

A.4.2 How to Add a Binding Template

By default, publishing a web service in OSR creates a binding of type HTTP. This is sufficient for the 10.1.3 version of Oracle JDeveloper to browse for the WSDL. However, for release 11g R1, you must create an additional binding template of type **wsdlDeployment**. The SOA Infrastructure uses the orauddi protocol to communicate with OSR and retrieve the WSDL URL.

To add a binding template

1. Right-click the business service to publish, and select **Add Binding**.
2. Add the access point (for example, `http://hostname:port/Proj_ep?WSDL`). The access point is the service WSDL URL.
3. From the **Use Type** list, select **wsdlDeployment**.
4. Click **Add Binding**.
5. Click **Save**. The service is now published to the registry and the orauddi protocol is configured.

A.4.3 How to Create a Connection to the Registry

To create a connection to the registry:

1. Go to Oracle JDeveloper.
2. Select **File > New > Connections > UDDI Registry Connection** to create a UDDI connection.
3. Enter a connection name.
4. Enter an inquiry endpoint URL. For example:
`http://myhost.us.oracle.com:7001/registry/uddi/inquiry`
5. Ensure that the **Business View** radio button is selected.
6. Click **Next**.
7. Click **Test Connection**.
8. If successful, click **Finish**. Otherwise, click the **Back** button and correct your errors.

A.4.4 How to Configure a SOA project to Invoke a Service from the Registry

To configure a SOA project to invoke a service from the registry:

1. Open the SOA project in which to create a reference to the business service.
2. Drag a **Web Service** icon into the **External Services** swimlane.
The Create Web Service dialog appears.
3. To the right of the **WSDL URL** field, click the first icon to select a WSDL.
4. From the list at the top, select **Resource Palette**.

5. Expand the navigational tree.
6. Expand **UDDI Registry > Business Services**.
7. Select the published business service, and click **OK**.

Note: If a no `wsdlDeployment` binding in UDDI message is displayed, it means that the additional binding template was not added as described in [Section A.4.2, "How to Add a Binding Template."](#)

8. Complete the remaining fields in the Create Web Service dialog, and click **OK**.
9. Wire the reference with the appropriate service component.
10. Perform additional modeling, as necessary.
11. In the SOA Composite Editor, click **Source**.

The `composite.xml` file stores the `serviceKey` within the `orauddi` protocol tag. The endpoint URL is not stored in the SOA composite application. At the time of SOA Infrastructure restart, the `orauddi` key is resolved and the endpoint URL is retrieved from the registry. Each WSDL has its own unique service key.

```
<property name="oracle.soa.uddi.serviceKey" type="xs:string"
  many="false">uddi:bc2785c0-350c-11de-94cb-1c7f0d2094c6</property>
```

A.4.5 How To Configure the Inquiry URL for Runtime

To configure the inquiry URL for runtime:

1. Log in to Oracle Enterprise Manager Fusion Middleware Control Console.
2. From the **SOA Infrastructure** menu, select **SOA Administration > Common Properties**.
3. Specify the same UDDI inquiry URL as you specified in the Create UDDI Registry Connection wizard:

```
http://myhost.us.oracle.com:7001/registry/uddi/inquiry
```

Note: Do *not* enter a user name or password. These are only used for the secure HTTP configuration of OSR.

4. Click **Apply**.
5. Exit Oracle Enterprise Manager Fusion Middleware Control Console.
6. To see endpoint statistics, return to the Registry Control.
7. Go to the Manage page and check statistics to see the increase in the number of invocations when not cached (the first time).

Caching of endpoint WSDL URLs occurs by default during runtime. If an endpoint WSDL URL is resolved using the `orauddi` protocol, subsequent invocations retrieve the WSDL URLs from cache, and not from OSR. When an endpoint WSDL obtained from cache is no longer reachable, the cache is refreshed and OSR is contacted to retrieve the new endpoint WSDL location. As a best practice, Oracle recommends that you undeploy services that are no longer required in Oracle Enterprise Manager Fusion Middleware Control Console and

used by the SOA Infrastructure. Endpoint services that are shut down or retired (but not undeployed) are still reachable. Therefore, the cache is not refreshed.

If you move the business service WSDL from one host to another, ensure that you change the location in the Registry Control. No change is required in Oracle JDeveloper or Oracle Enterprise Manager Fusion Middleware Control Console.

Note: In 11g, caching occurs automatically. If you are using Oracle SOA Suite 10.1.3, caching is supported by setting the `CacheRegistryWSDL` property to `true` in `bpel.xml`. Setting this property to `false` disables caching.

A.5 Validating When Loading a Process Diagram

You may see an icon (a yellow triangle with an exclamation point) indicating invalid settings as you create and open activities such as a scope or an assign for the first time. The settings are invalid because you have not yet entered details.

To turn this option off for the current project, do the following:

1. Right-click the BPEL diagram and select **Display > Diagram Properties**.
2. Deselect the **Enable Automatic Validation** option.
3. Click **OK**.
4. Select **Save All** from the **File** main menu.

XPath Extension Functions

This appendix describes the XPath extension functions. Oracle provides XPath functions that use the capabilities built into Oracle SOA Suite and XPath standards for adding new functions.

This appendix includes the following sections:

- [Section B.1, "SOA XPath Extension Functions"](#)
- [Section B.2, "BPEL XPath Extension Functions"](#)
- [Section B.3, "Mediator XPath Extension Functions"](#)
- [Section B.4, "Advanced Functions"](#)
- [Section B.5, "Workflow Service Functions"](#)
- [Section B.6, "Using the XPath Building Assistant"](#)
- [Section B.7, "Creating User-Defined XPath Extension Functions"](#)

For additional information about XPath functions, visit the following URL:

<http://www.w3.org>

B.1 SOA XPath Extension Functions

This section describes the following SOA XPath extension functions:

- [Section B.1.1, "Database Functions"](#)
- [Section B.1.2, "Date Functions"](#)
- [Section B.1.3, "Mathematical Functions"](#)
- [Section B.1.4, "String Functions"](#)

B.1.1 Database Functions

This section describes the following database functions:

B.1.1.1 lookup-table

This function returns a string based on the SQL query generated from the parameters.

The string is obtained by executing:

```
SELECT outputColumn FROM table WHERE inputColumn = key
```

against the data source that can be either a JDBC connect string

(`jdbc:oracle:thin:username/password@host:port:sid`) or a data source

JNDI identifier. Only the Oracle Thin Driver is supported if the JDBC connect string is used.

Example: `oraext:lookup-table('employee', 'id', '1234', 'last_name', 'jdbc:oracle:thin:xyz/xyz@localhost:1521:ORCL')`

Signature:

`oraext:lookup-table(table, inputColumn, key, outputColumn, data source)`

Arguments:

- `table` - The table from which to draw the data.
- `inputColumn` - The column within the table.
- `key` - The key value of the input column.
- `outputColumn` - The column to output the data.
- `data source` - The source of the data.

Property IDs:

- `namespace-uri:`
`http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.ExtFunc`
- `namespace-prefix:` `oraext`

B.1.1.2 query-database

This function returns a node set by executing the SQL query against the specified database.

Signature:

`oraext:query-database(sqlquery as string, rowset as boolean, row as boolean, data source as string)`

Arguments:

- `sqlquery` - The SQL query to perform.
- `rowset` - Indicates if the rows should be enclosed in an element.
- `row` - Indicates if each row should be enclosed in an element.
- `data source` - Either a JDBC connect string (`jdbc:oracle:thin:username/password@host:port:sid`) or a JNDI name for the database.

Property IDs:

- `namespace-uri:`
`http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.ExtFunc`
- `namespace-prefix:` `oraext`

B.1.1.3 sequence-next-val

Returns the next value of an Oracle sequence.

The next value is obtained by executing

```
SELECT sequence.nextval FROM dual
```

against a data source that can be either a JDBC connect string (`jdbc:oracle:thin:username/password@host:port:sid`) or a data source JNDI identifier. Only the Oracle Thin Driver is supported if a JDBC connect string is used.

Example: `oraext:sequence-next-val('employee_id_sequence', 'jdbc:oracle:thin:xyz/xyz@localhost:1521:ORCL')`

Signature:

`oraext:sequence-next-val(sequence as string, data source as string)`

Arguments:

- `sequence` - The sequence number in the database.
- `data source` - Either a JDBC connect string or a data source JNDI identifier.

Property IDs:

- `namespace-uri:`
`http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.ExtFunc`
- `namespace-prefix:` `oraext`

B.1.2 Date Functions

This section describes the following functions:

B.1.2.1 add-dayTimeDuration-to-dateTime

This function returns a new date time value adding `dateTime` to the given duration.

If the duration value is negative, then the resulting value precedes `dateTime`.

Signature:

`xpath20:add-dayTimeDuration-from-dateTime(dateTime as string, duration as string)`

Arguments:

- `dateTime as string` - The `dateTime` to which the function adds the duration, in string format.
- `duration as string` - The duration to add to the `dateTime`, or subtract if the duration is negative, in string format.

Property IDs:

- `namespace-uri:`
`http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.XPath20`
- `namespace-prefix:` `xpath20`

B.1.2.2 current-date

This function returns the current date in ISO format `YYYY-MM-DD`.

Signature:

`xpath20:current-date(object)`

Arguments:

- `object` - The time in standard format

Property IDs:

- `namespace-uri:`
`http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.XPath20`
- `namespace-prefix:` `xpath20`

B.1.2.3 current-dateTime

This function returns the current datetime value in ISO format `CCYY-MM-DDThh:mm:ssTZD`.

Signature:

`xpath20:current-dateTime(object)`

Arguments:

- `object` - The time in standard format

Property IDs:

- `namespace-uri:`
`http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.XPath20`
- `namespace-prefix:` `xpath20`

B.1.2.4 current-time

This function returns the current time in ISO format. The format is `hh:mm:ssTZD`.

Signature:

`xpath20:current-time(object)`

Arguments:

- `object` - The time in standard format

Property IDs:

- `namespace-uri:`
`http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.XPath20`
- `namespace-prefix:` `xpath20`

B.1.2.5 day-from-dateTime

This function returns the day from `dateTime`. The default day is 1.

Signature:

`xpath20:day-from-dateTime(object)`

Arguments:

- `object` - The time in standard format as a string.

Property IDs:

- namespace-uri:
http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.XPath20
- namespace-prefix: xpath20

B.1.2.6 format-dateTime

This function returns the formatted string of dateTime using the format provided.

Signature:

xpath20:format-dateTime(dateTime as string, format as string)

Arguments:

- dateTime - The dateTime to be formatted.
- format - The format for the output.

Property IDs:

- namespace-uri:
http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.XPath20
- namespace-prefix: xpath20

B.1.2.7 hours-from-dateTime

This function returns the hour from dateTime. The default hour is 0.

Signature:

xpath20:hours-from-dateTime(dateTime as string)

Arguments:

- dateTime - The string with the date and time.

Property IDs:

- namespace-uri:
http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.XPath20
- namespace-prefix: xpath20

B.1.2.8 implicit-timezone

This function returns the current time zone in ISO format +/- hh:mm, indicating a deviation from UTC (Coordinated Universal Timezone).

Signature:

xpath20:implicit-timezone(object)

Arguments:

- object - The time in standard format.

Property IDs:

- namespace-uri:
http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.XPath20
- namespace-prefix: xpath20

B.1.2.9 minutes-from-dateTime

This function returns the minute from `dateTime`. The default minute is 0.

Signature:

```
xpath20:minutes-from-dateTime(dateTime as string)
```

Arguments:

- `dateTime as string` - The date and time.

Property IDs:

- `namespace-uri:`
`http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.XPath20`
- `namespace-prefix:` `xpath20`

B.1.2.10 month-from-dateTime

This function returns the month from `dateTime`. The default month is 1 (January).

Signature:

```
xpath20:month-from-dateTime(dateTime as string)
```

Arguments:

- `dateTime as string` - The `dateTime` to be formatted.

Property IDs:

- `namespace-uri:`
`http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.XPath20`
- `namespace-prefix:` `xpath20`

B.1.2.11 seconds-from-dateTime

This function returns the second from `dateTime`. The default second is 0.

Signature:

```
xpath20:seconds-from-dateTime(dateTime as string)
```

Arguments:

- `dateTime as a string` - The `dateTime` as a string.

Property IDs:

- `namespace-uri:`
`http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.XPath20`
- `namespace-prefix:` `xpath20`

B.1.2.12 subtract-dayTimeDuration-from-dateTime

This function returns a new `dateTime` value after subtracting the duration from `dateTime`.

If the duration value is negative, then the resultant `dateTime` value follows `input-dateTime` value.

Signature:

`xpath20:subtract-dayTimeDuration-from-dateTime(dateTime as string, duration as string)`

Arguments:

- `dateTime as string` - The `dateTime` from which the function subtracts the duration, in string format.
- `duration as string` - The duration to subtract to the `dateTime`, or to add if the duration is negative, in string format.

Property IDs:

- `namespace-uri:`
`http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.XPath20`
- `namespace-prefix:` `xp20`

B.1.2.13 `timezone-from-dateTime`

This function returns the time zone from `dateTime`. The default time zone is GMT+00:00.

Signature:

`xpath20:timezone-from-dateTime(dateTime as string)`

Arguments:

- `dateTime as string` - The `dateTime` for which this function returns a time zone.

Property IDs:

- `namespace-uri:`
`http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.XPath20`
- `namespace-prefix:` `xpath20`

B.1.2.14 `year-from-dateTime`

This function returns the year from `dateTime`.

Signature:

`xpath20:year-from-dateTime(dateTime as string)`

Arguments:

- `dateTime` - The `dateTime` as a string.

Property IDs:

- `namespace-uri:`
`http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.XPath20`
- `namespace-prefix:` `xpath20`

B.1.3 Mathematical Functions

This section describes the following function.

B.1.3.1 abs

This function returns the absolute value of `inputNumber`.

If `inputNumber` is not negative, the `inputNumber` is returned. If the `inputNumber` is negative, the negation of `inputNumber` is returned.

Example: `abs (-1)` returns 1.

Signature:

```
xpath20:abs(inputNumber as number)
```

Arguments:

- `inputNumber as number` - The number for which the function returns an absolute value.

Property IDs:

- `namespace-uri:`
`http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.XPath20`
- `namespace-prefix:` `xpath20`

B.1.4 String Functions

This section describes the string functions.

B.1.4.1 compare

This function returns the lexicographical difference between `inputString` and `compareString` by comparing the unicode value of each character of both the strings.

This function returns -1 if `inputString` lexicographically precedes the `compareString`.

This function returns 0 if both `inputString` and `compareString` are equal.

This function returns 1 if `inputString` lexicographically follows the `compareString`.

Example: `xpath20:compare ('Audi ', 'BMW')` returns -1

Signature:

```
xpath20:compare(inputString as string, compareString as string)
```

Arguments:

- `variableName` - The source variable for the data.
- `propertyName` - The qualified name (QName) of the property.

Property IDs:

- `namespace-uri:`
`http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.XPath20`
- `namespace-prefix:` `xpath20`

B.1.4.2 compare-ignore-case

This function returns the lexicographical difference between `inputString` and `compareString` while ignoring case and comparing the unicode value of each character of both the strings.

This function returns -1 if `inputString` lexicographically precedes the `compareString`.

This function returns 0 if both `inputString` and `compareString` are equal.

This function returns 1 if `inputString` lexicographically follows the `compareString`.

Example: `xpath20:compare-ignore-case('Audi', 'bmw')` returns -1

Signature:

```
xp:compare-ignore-case(inputString as string, compareString as string)
```

Arguments:

- `inputString` - The string of data to be searched.
- `CompareString` - The string to compare against the input string.

Property IDs:

- `namespace-uri:`
`http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.Xpath20`
- `namespace-prefix:` `xpath20`

B.1.4.3 create-delimited-string

This function returns a delimited string created from `nodeSet` delimited by `delimiter`.

Signature:

```
oraext:create-delimited-string(nodeSet as node-set, delimiter as string)
```

Arguments:

- `nodeSet` - The node set to be converted into a delimited string.
- `delimiter` - The character that separates the items in the output string; for example, a comma or a semicolon.

Property IDs:

- `namespace-uri:`
`http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.ExtFunc`
- `namespace-prefix:` `oraext`

B.1.4.4 ends-with

This function returns true if `inputString` ends with `searchString`.

Example: `xpath20:ends-with('XSL Map', 'Map')` returns true

Signature:

```
xpath20:ends-with(inputString as string, searchString as string)
```

Arguments:

- `inputString` - The string of data to be searched.
- `searchString` - The string for which the function searches.

Property IDs:

- `namespace-uri`:
`http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.XPath20`
- `namespace-prefix`: `xpath20`

B.1.4.5 format-string

This function returns the message formatted with the arguments passed. At least one argument is required and supports up to a maximum of 10 arguments.

Example: `oraext:format-string('{0} + {1} = {2}', '2', '2', '4')`
returns `'2 + 2 = 4'`

Signature:

`oraext:format-string(string, string, string...)`

Arguments:

- `string` - One of the strings to be used in the formatted output.

Property IDs:

- `namespace-uri`:
`http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.ExtFunc`
- `namespace-prefix`: `oraext`

B.1.4.6 get-content-as-string

This function returns the XML representation of the input element.

Signature:

`oraext:get-content-as-string(element as node-set)`

Arguments:

- `element as node-set` - The input element that the function returns as an XML representation.

Property IDs:

- `namespace-uri`:
`http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.ExtFunc`
- `namespace-prefix`: `oraext`

B.1.4.7 get-content-from-file-function

This function returns the content of the file.

Signature:

`oraext:get-content-from-file-function(object)`

Arguments:

- object:

Property IDs:

- namespace-uri:
http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.ExtFunc
- namespace-prefix: oraext

B.1.4.8 get-localized-string

This function returns the locale-specific string for the key. This function uses language, country, variant, and resource bundle to identify the correct resource bundle.

The resource bundle is obtained by resolving resourceLocation against the resourceBaseURL. The URL is assumed to be a directory only if it ends with /.

Usage: oraext:get-localized-string(resourceBaseURL as string, resourceLocation as string, resource bundle as string, language as string, country as string, variant as string, key as string)

Example:

oraext:get-localized-string('file:/c:/', '', 'MyResourceBundle', 'en', 'US', '', 'MSG_KEY') returns a locale-specific string from a resource bundle 'MyResourceBundle' in the C:\ directory

Signature:

oraext:get-localized-string(resourceURL, resourceLocation, resourceBundleName, language, country, variant, messageKey)

Arguments:

- resourceURL - The URL of the resource.
- resourceLocation - The subdirectory location of the resource.
- resourceBundleName - The name of the ZIP file containing the resource bundle.
- language - The language of the localized output.
- country - The country of the localized output.
- variant - The language variant of the localized output.
- messageKey - The message key in the resource bundle.

Property IDs:

- namespace-uri:
http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.ExtFunc
- namespace-prefix: oraext

B.1.4.9 index-within-string

This function returns the zero-based index of the first occurrence of searchString within the inputString.

This function returns -1 if searchString is not found.

Example: oraext:index-within-string('ABCABC', 'B') returns 1

Signature:

```
oraext:index-within-string(inputString as string, searchString  
as string)
```

Arguments:

- `inputString` - The string of data to be searched.
- `searchString` - The string for which the function searches in `inputString`.

Property IDs:

- `namespace-uri:`
`http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.ExtFunc`
- `namespace-prefix:` `oraext`

B.1.4.10 last-index-within-string

This function returns the zero-based index of the last occurrence of `searchString` within `inputString`.

This function returns -1 if `searchString` is not found.

Example: `oraext:last-index-within-string('ABCABC', 'B')` returns 4

Signature:

```
oraext:last-index-within-string(inputString as string,  
searchString as string)
```

Arguments:

- `inputString` - The string of data to be searched.
- `searchString` - The string for which the function searches in the `inputString`.

Property IDs:

- `namespace-uri:`
`http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.ExtFunc`
- `namespace-prefix:` `oraext`

B.1.4.11 left-trim

This function returns the value of `inputString` after removing all the leading white spaces.

Example: `oraext:left-trim(' account ')` returns `'account '`

Signature:

```
oraext:left-trim(inputString)
```

Arguments:

- `inputString` - The string to be left-trimmed.

Property IDs:

- `namespace-uri:`
`http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.ExtFunc`
- `namespace-prefix:` `oraext`

B.1.4.12 lower-case

This function returns the value of `inputString` after translating every character to its lower-case correspondent.

Example: `xpath20:lower-case('ABc!D')` returns `'abc!d'`

Signature:

```
xpath20:lower-case(inputString)
```

Arguments:

- `inputString` - The string of data that is in lowercase.

Property IDs:

- `namespace-uri:`
`http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.XPath20`
- `namespace-prefix:` `xpath20`

B.1.4.13 matches

This function returns true if `inputString` matches the regular expression pattern `regexPattern`.

Example: `xpath20:matches('abracadabra', '^a.*a$')` returns true

Signature:

```
xpath20:matches(inputString, regexPattern)
```

Arguments:

- `inputString` - The string of data that must be matched.
- `regexPattern` - The regular expression pattern.

Property IDs:

- `namespace-uri:`
`http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.XPath20`
- `namespace-prefix:` `xpath20`

B.1.4.14 right-trim

This function returns the value `inputString` after removing all the trailing white spaces.

Example: `oraext:right-trim(' account ')` returns `' account '`

Signature:

```
oraext:right-trim(inputString as string)
```

Arguments:

- `inputString` - The input string to be right-trimmed.

Property IDs:

- `namespace-uri:`
`http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.ExtFunc`

- namespace-prefix: oraext

B.1.4.15 upper-case

This function returns the value of `inputString` after translating every character to its uppercase correspondent.

Example: `xpath20:upper-case('abCd0')` returns 'ABCD0'

Signature:

```
xpath20:upper-case(inputString as string)
```

Arguments:

- `inputString` - The string of data that is in uppercase.

Property IDs:

- namespace-uri:
`http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.XPath20`
- namespace-prefix: xpath20

B.2 BPEL XPath Extension Functions

This section describes the following BPEL XPath extension functions:

B.2.1 addQuotes

This function returns the content of a `string` with single quotes added.

Signature:

```
ora:addQuotes(string)
```

Arguments:

- `string` - The string to which this function adds quotes.

Property IDs:

- namespace-uri: `http://schemas.oracle.com/xpath/extension`
- namespace-prefix: ora

B.2.2 appendToList

Note: The `appendToList` function is deprecated. Oracle recommends that you use the `bpelx:copyList` extension of an `assign` activity to append data to a node list.

This function appends to a node list. The node list with which to append should not be null or empty.

Signature:

```
ora:appendToList('variableName', 'partName'?, 'locationPath'?,  
Object)
```

Arguments:

- `variableName` - The source variable for the data.
- `partName` - The part to select from the variable (optional).
- `locationPath` - Provides an absolute location path (with / meaning the root of the document fragment representing the entire part) to identify the root of a subtree within the document fragment representing the part (optional).
- `Object` - The object can be either a list or a single item. If the object is a list, this function appends each item in the list. Each appended item is either an element, or an element with the string value of the node created.

Property IDs:

- `deprecated`
Use the `bpelx:copyList` or `bpelx:append` extension activity to append to a list.
- `namespace-uri: http://schemas.oracle.com/xpath/extension`
- `namespace-prefix: ora`

B.2.3 copyList

Note: While the `copyList` function is still available for use, Oracle recommends that you use the `bpelx:copyList` extension to copy a node list or a node.

This function copies a node list or a node. The node list to be copied to should not be null or empty.

Signature:

```
ora:copyList('variableName', 'partName?', 'locationPath?',
Object)
```

Arguments:

- `variableName` - The source variable for the data.
- `partName` - The part to select from the variable (optional).
- `locationPath` - Provides an absolute location path (with / meaning the root of the document fragment representing the entire part) to identify the root of a subtree within the document fragment representing the part (optional).
- `Object` - The object can be either a list or a single item. If the object is a list, each item in the list is copied. Each item to be copied is either an element, or an element with the string value of the node created.

Property IDs:

- `deprecated`
Use the `bpelx:copyList` extension activity to append to a list.
- `namespace-uri: http://schemas.oracle.com/xpath/extension`
- `namespace-prefix: ora`

B.2.4 countNodes

Note: While the `countNodes` function is still available for use, Oracle recommends that you use version 1.0 of the XPath `count()` function to return the size of the elements as an integer.

This function returns the size of the elements as an integer.

Signature:

```
ora:countNodes('variableName', 'partName'?, 'locationPath'?)
```

Arguments:

- `variableName` - The source variable for the data.
- `partName` - The part to select from the variable (optional).
- `locationPath` - Provides an absolute location path (with / meaning the root of the document fragment representing the entire part) to identify the root of a subtree within the document fragment representing the part (optional).

Property IDs:

- `namespace-uri`: `http://schemas.oracle.com/xpath/extension`
- `namespace-prefix`: `ora`

B.2.5 doc

This function returns the content of an XML file.

Signature:

```
ora:doc('fileName', 'xpath'?)
```

Arguments:

- `fileName` - The name of the XML file.
- `xpath` - The path to the file.

Property IDs:

- `namespace-uri`: `http://schemas.oracle.com/xpath/extension`
- `namespace-prefix`: `ora`

B.2.6 doStreamingTranslate

This function translates using the streaming XPath APIs. It uses a unique concept called batching so that the transformation engine does not materialize the result of transformation into memory. Therefore, it can handle arbitrarily large payloads of the order of gigabytes. However, it can handle only forward-only XSL constructs such as `for-each`. The `targetType` can be `SDOM` or `ATTACHMENT`.

Signature:

```
ora:doStreamingTranslate('input SDOM or attachment element',  
'streaming xpath context', 'SDOM or ATTACHMENT', 'attachment  
element?')
```

Arguments:

- input SDOM or attachment element
- streaming xpath context
- SDOM or ATTACHMENT
- attachment element

Property IDs:

- namespace-uri: http://schemas.oracle.com/xpath/extension
- namespace-prefix: ora

B.2.7 doTranslateFromNative

This function translates the input data to XML, where the input can be a string, attachment, or element that contains Base64-encoded data. The `targetType` can be DOM, ATTACHMENT or SDOM.

Signature:

```
ora:doTranslateFromNative('input', 'nxsdTemplate', 'nxsdRoot', 'targetType', 'attachment element?')
```

Arguments:

- input - The input data of the XPath function.
- nxsdTemplate - The NXSD schema that you want to use to translate the input data to XML format.
- nxsdRoot - The root element in the NXSD schema.
- targetType - Decides how the XPath function translates the native data into XML.
- attachment element - This is the attachment for the returned XML. This parameter is optional.

Property IDs:

- namespace-uri: http://schemas.oracle.com/xpath/extension
- namespace-prefix: ora

B.2.8 doTranslateToNative

This function translates the input DOM to a string or attachment. The `targetType` can be STRING or ATTACHMENT

Signature:

```
ora:doTranslateToNative('input', 'nxsdTemplate', 'nxsdRoot', 'targetType', 'attachment element?')
```

Arguments:

- input - The input data of the XPath function.
- nxsdTemplate - The NXSD schema that you want to use to translate the input data to XML format.
- nxsdRoot - The root element in the NXSD schema.
- targetType - Decides how the XPath function translates the native data into XML.

- `attachment element` - This is the attachment for the returned XML. This parameter is optional.

Property IDs:

- `namespace-uri`: `http://schemas.oracle.com/xpath/extension`
- `namespace-prefix`: `ora`

B.2.9 doXSLTransform

This function implements WS-BPEL 2.0's `doXSLTransform` function that supports multiple parameters of XSLT. When using this function, the XSL template match must not be set to `root` (which is `/`). It must be the root element.

Signature:

```
ora:doXSLTransform('url_to_xslt', input, ['paramQname', paramValue]*)
```

Arguments:

- `url_to_xslt` - Specifies the XSL style sheet URL.
- `input` - Specifies the input variable name.
- `paramQname` - Specifies the parameter QName.
- `paramValue` - Specifies the value of the parameter.

Property IDs:

- `namespace-uri`: `http://schemas.oracle.com/xpath/extension`
- `namespace-prefix`: `ora`

B.2.10 doXSLTransformForDoc

This function is a complement XPath function to `doXSLTransform()`. It aims to perform the transformation when the XSLT template matches the document.

Signature:

```
ora:doXSLTransformForDoc('url_to_xslt', input, ['paramQname', paramValue]*)
```

Arguments:

- `url_to_xslt` - Specifies the XSL style sheet URL.
- `input` - Specifies the input variable name.
- `paramQname` - Specifies the parameter QName.
- `paramValue` - Specifies the value of the parameter.

Property IDs:

- `namespace-uri`: `http://schemas.oracle.com/xpath/extension`
- `namespace-prefix`: `ora`

B.2.11 formatDate

This function converts standard XSD date formats to characters suitable for output.

Signature:

```
ora:formatDate('dateTime', 'format')
```

Arguments:

- `dateTime` - Contains a date-related value in XSD format. For nonstring arguments, this function behaves as if a `string()` function were applied. If the argument is not a date, the output is an empty string. If it is a valid XSD date and some fields are empty, this function attempts to fill unspecified fields. For example, `2003-06-10T15:56:00`.
- `format` - Contains a string formatted according to `java.text.SimpleDateFormat` format

Property IDs:

- `namespace-uri`: `http://schemas.oracle.com/xpath/extension`
- `namespace-prefix`: `ora`

B.2.12 generateGUID

Generates a unique GUID.

Signature:

```
ora:generateGUID()
```

Arguments: There are no arguments for this function.

Property IDs:

- `namespace-uri`: `http://schemas.oracle.com/xpath/extension`
- `namespace-prefix`: `ora`

B.2.13 getApplicationName

This function returns the application name.

Signature:

```
ora:getApplicationName()
```

Arguments: There are no arguments for this function.

Property IDs:

- `namespace-uri`: `http://schemas.oracle.com/xpath/extension`
- `namespace-prefix`: `ora`

B.2.14 getAttachmentContent

This function gets the attachment content from an href function.

Signature:

```
ora:getAttachmentContent(varName[, partName[, query]])
```

Arguments:

- `varName` - Specifies the source variable for the data.
- `partName` - (Optional) Specifies the part to select from the variable.

- `query` - (Optional) Specifies an absolute location path (with `/` meaning the root of the document fragment representing the entire part) to identify the root of a subtree within the document fragment representing the part.

Property IDs:

- `namespace-uri`: `http://schemas.oracle.com/xpath/extension`
- `namespace-prefix`: `ora`

B.2.15 `getComponentName`

This function returns the component name.

Signature:

```
ora:getComponentName()
```

Arguments: There are no arguments for this function.

Property IDs:

- `namespace-uri`: `http://schemas.oracle.com/xpath/extension`
- `namespace-prefix`: `ora`

B.2.16 `getComponentInstanceID`

This function returns the component instance ID.

Signature:

```
ora:getComponentInstanceID()
```

Arguments: There are no arguments for this function.

Property IDs:

- `namespace-uri`: `http://schemas.oracle.com/xpath/extension`
- `namespace-prefix`: `ora`

B.2.17 `getCompositeName`

This function returns the composite name.

Signature:

```
ora:getCompositeName()
```

Arguments: There are no arguments for this function.

Property IDs:

- `namespace-uri`: `http://schemas.oracle.com/xpath/extension`
- `namespace-prefix`: `ora`

B.2.18 `getCompositeInstanceID`

This function returns the BPEL process composite instance ID.

Signature:

```
ora:getCompositeInstanceID()
```

Arguments: There are no arguments for this function.

Property IDs:

- namespace-uri: `http://schemas.oracle.com/xpath/extension`
- namespace-prefix: `ora`

B.2.19 getCompositeURL

This function returns the composite URL.

Signature:

```
ora:getCompositeURL()
```

Arguments: There are no arguments for this function.

Property IDs:

- namespace-uri: `http://schemas.oracle.com/xpath/extension`
- namespace-prefix: `ora`

B.2.20 getContentAsString

This function returns the content of an element as an XML string.

Signature:

```
ora:getContentAsString(element elementAsNodeList)
```

Arguments:

- `element` - The element (source of the data).
- `elementAsNodeList` - The element as the node list.

Property IDs:

- namespace-uri: `http://schemas.oracle.com/xpath/extension`
- namespace-prefix: `ora`

B.2.21 getConversationId

This function returns the conversation ID.

Signature:

```
ora:getConversationId()
```

Arguments: There are no arguments for this function.

Property IDs:

- namespace-uri: `http://schemas.oracle.com/xpath/extension`
- namespace-prefix: `ora`

B.2.22 getCreator

This function returns the instance creator.

Signature:

```
ora:getCreator()
```

Arguments: There are no arguments for this function.

Property IDs:

- namespace-uri: `http://schemas.oracle.com/xpath/extension`
- namespace-prefix: `ora`

B.2.23 `getCurrentDate`

This function returns the current date as a string.

Signature:

```
ora:getCurrentDate('format'?)
```

Argument:

- `format` - (Optional) Specifies a string formatted according to `java.text.SimpleDateFormat` `format` (optional).

Property IDs:

- namespace-uri: `http://schemas.oracle.com/xpath/extension`
- namespace-prefix: `ora`

B.2.24 `getCurrentDateTime`

This function returns the current date time as a string.

Signature:

```
ora:getCurrentDateTime('format'?)
```

Argument:

- `format` - (Optional) Specifies a string formatted according to `java.text.SimpleDateFormat` `format` (optional).

Property IDs:

- namespace-uri: `http://schemas.oracle.com/xpath/extension`
- namespace-prefix: `ora`

B.2.25 `currentTime`

This function returns the current time as a string.

Signature:

```
ora:currentTime('format'?)
```

Argument:

- `format` - (Optional) Specifies a string formatted according to `java.text.SimpleDateFormat` `format` (optional).

Property IDs:

- namespace-uri: `http://schemas.oracle.com/xpath/extension`
- namespace-prefix: `ora`

B.2.26 `getDomainId`

This function returns the current domain ID.

Signature:

```
ora:getDomainId()
```

Arguments: There are no arguments for this function.

Property IDs:

- namespace-uri: `http://schemas.oracle.com/xpath/extension`
- namespace-prefix: `ora`

B.2.27 getECID

This function returns ECID.

Signature:

```
ora:getECID()
```

Arguments: There are no arguments for this function.

Property IDs:

- namespace-uri: `http://schemas.oracle.com/xpath/extension`
- namespace-prefix: `ora`

B.2.28 getElement

This function returns an element using `index` from the array of elements.

Signature:

```
ora:getElement('variableName', 'partName', 'locationPath',  
index)
```

Arguments:

- `variableName` - The source variable for the data.
- `partName` - The part to select from the variable (required).
- `locationPath` - Provides an absolute location path (with / meaning the root of the document fragment representing the entire part) to identify the root of a subtree within the document fragment representing the part (required).
- `index` - Dynamic index value. The index of the first node is 1.

Property IDs:

- namespace-uri: `http://schemas.oracle.com/xpath/extension`
- namespace-prefix: `ora`

B.2.29 getFaultAsString

This function returns the fault as a string value.

Signature:

```
ora:getFaultAsString()
```

Arguments: There are no arguments for this function.

Property IDs:

- namespace-uri: `http://schemas.oracle.com/xpath/extension`
- namespace-prefix: `ora`

B.2.30 getFaultName

This function returns the fault name.

Signature:

```
ora:getFaultName()
```

Arguments: There are no arguments for this function.

Property IDs:

- namespace-uri: `http://schemas.oracle.com/xpath/extension`
- namespace-prefix: `ora`

B.2.31 getGroupIdsFromGroupAlias

This function returns a List of user Ids for a group alias specified in the TaskServiceAliases section of the BPEL suitcase descriptor.

Signature:

```
ora:getGroupIdsFromGroupAlias(String aliasName)
```

Arguments:

- aliasName - The alias for a list of users or groups.

Property IDs:

- namespace-uri: `http://schemas.oracle.com/xpath/extension`
- namespace-prefix: `ora`

B.2.32 getInstanceId

This function returns the instance ID.

Signature:

```
ora:getInstanceId()
```

Arguments: There are no arguments for this function.

Property IDs:

- namespace-uri: `http://schemas.oracle.com/xpath/extension`
- namespace-prefix: `ora`

B.2.33 getNodeValue

This function returns the value of a DOM node as a string.

Signature:

```
ora:getNodeValue(node)
```

Arguments:

- node - The DOM node.

Property IDs:

- namespace-uri: `http://schemas.oracle.com/xpath/extension`
- namespace-prefix: `ora`

B.2.34 `getNodes`

This function gets a node list. This is implemented as an alternate to `bpws:getVariableData`, which does not return a node list.

Signature:

```
ora:getNodes('variableName', 'partName'?, 'locationPath'?)
```

Arguments:

- `variableName` - The source variable for the data.
- `partName` - The part to select from the variable (optional).
- `locationPath` - Provides an absolute location path (with / meaning the root of the document fragment representing the entire part) to identify the root of a subtree within the document fragment representing the part (optional).

Property IDs:

- `namespace-uri`: `http://schemas.oracle.com/xpath/extension`
- `namespace-prefix`: `ora`

B.2.35 `getOwnerDocument`

This function returns the document object associated with the node.

Signature:

```
ora:getOwnerDocument (node)
```

Arguments:

- `node` - Specifies the XML node.

Property IDs:

- `namespace-uri`: `http://schemas.oracle.com/xpath/extension`
- `namespace-prefix`: `ora`

B.2.36 `getParentComponentInstanceID`

This function returns the BPEL process instance parent component instance ID.

Signature:

```
ora:getParentComponentInstanceID()
```

Arguments: There are no arguments for this function.

Property IDs:

- `namespace-uri`: `http://schemas.oracle.com/xpath/extension`
- `namespace-prefix`: `ora`

B.2.37 `getPreference`

This function returns the value of a property specified in the preferences section of the BPEL suitcase descriptor.

Signature:

```
ora:getPreference (preferenceName)
```

Arguments:

- `preferenceName` - The name of the preference as specified in the BPEL suitcase descriptor.

Property IDs:

- `namespace-uri: http://schemas.oracle.com/xpath/extension`
- `namespace-prefix: ora`

B.2.38 getProcessId

This function returns the ID of the current BPEL process.

Signature:

```
ora:getProcessId()
```

Arguments: There are no arguments for this function.

Property IDs:

- `namespace-uri: http://schemas.oracle.com/xpath/extension`
- `namespace-prefix: ora`

B.2.39 getProcessOwnerId

This function returns the ID of the user who owns the process, if specified in the `TaskServiceAliases` section of the BPEL suitcase descriptor.

Signature:

```
ora:getProcessOwnerId()
```

Arguments: There are no arguments for this function.

Property IDs:

- `namespace-uri: http://schemas.oracle.com/xpath/extension`
- `namespace-prefix: ora`

B.2.40 getProcessURL

This function returns the root URL of the current BPEL process.

Signature:

```
ora:getProcessURL()
```

Arguments: There are no arguments for this function.

Property IDs:

- `namespace-uri: http://schemas.oracle.com/xpath/extension`
- `namespace-prefix: ora`

B.2.41 getProcessVersion

This function returns the current process version.

Signature:

```
ora:getProcessVersion()
```

Arguments: There are no arguments for this function.

Property IDs:

- namespace-uri: `http://schemas.oracle.com/xpath/extension`
- namespace-prefix: `ora`

B.2.42 `getUserAliasId`

This function returns the user ID for an alias specified in the `TaskServiceAliases` section of the BPEL suitcase descriptor.

Signature:

```
ora:getUserAliasId (String aliasName)
```

Arguments:

- `aliasName` - The alias for a list of users or groups.

Property IDs:

- namespace-uri: `http://schemas.oracle.com/xpath/extension`
- namespace-prefix: `ora`

B.2.43 `getUserIdsFromGroupAlias`

This function returns a List of user IDs for a group alias specified in the `TaskServiceAliases` section of the BPEL suitcase descriptor.

Signature:

```
ora:getUserIdsFromGroupAlias( String aliasName )
```

Arguments:

- `aliasName` - Alias name of the group.

Property IDs:

- namespace-uri: `http://schemas.oracle.com/xpath/extension`
- namespace-prefix: `ora`

B.2.44 `setCompositeInstanceTitle`

This function sets a title to the composite instance which can later be used as one of the criteria in searching the instances. This function returns the same string that is passed as the argument.

Signature:

```
med:setCompositeInstanceTitle(title)
```

Arguments:

- `title` - Specifies the composite instance title. This can be specified as an XPath expression on the message payload.

Property IDs:

- namespace-uri: `http://schemas.oracle.com/xpath/extension`
- namespace-prefix: `ora`

B.2.45 instanceOf

This function extracts arbitrary values from BPEL variables.

Signature:

```
ora:instanceOf(an_xpath_expression, 'typeQName')
```

Arguments:

- `an_xpath_expression` - An XPath expression that returns an element
- `typeQName` - The QName of a global declared XSD type

Property IDs:

- `namespace-uri: http://schemas.oracle.com/xpath/extension`
- `namespace-prefix: ora`

B.2.46 integer

This function returns the content of the node as an integer.

Signature:

```
ora:integer(node)
```

Arguments:

- `node` - The input node.

Property IDs:

- `namespace-uri: http://schemas.oracle.com/xpath/extension`
- `namespace-prefix: ora`

B.2.47 parseEscapedXML

This function parses a string to DOM.

Signature:

```
ora:parseEscapedXML(contentString)
```

Arguments:

- `contentString` - The string that this function parses to a DOM.

Property IDs:

- `namespace-uri: http://schemas.oracle.com/xpath/extension`
- `namespace-prefix: ora`

B.2.48 parseXML

This function parses a string to DOM.

Signature:

```
ora:parseXML(contentString)
```

Arguments:

- `contentString` - The string that this function parses to a DOM.

Property IDs:

- namespace-uri: `http://schemas.oracle.com/xpath/extension`
- namespace-prefix: `ora`

B.2.49 processXQuery

This function returns the result of an XQuery transformation.

Signature:

```
ora:ry('template','context?')
```

Arguments:

- `template` - The XSLT template.
- `input` - The input data to be transformed.
- `properties` - The properties as defined in the `bpel.xml` file.

Property IDs:

- namespace-uri: `http://schemas.oracle.com/xpath/extension`
- namespace-prefix: `ora`

B.2.50 processXSLT

This function returns the result of XSLT transformation using the Oracle XDK XSLT processor.

Signature:

```
xdk:processXSLT('template','input','properties?')
```

Arguments:

- `template` - The XSLT template.
- `input` - The input data to be transformed.
- `properties` - The properties as defined in the `bpel.xml` file.

Property IDs:

- namespace-uri: `http://schemas.oracle.com/xpath/extension`
- namespace-prefix: `xdk`

B.2.51 processXSLTAttachment

This function returns the results of XSLT transformation by using the Oracle XDK XSLT processor. This function also supports transformations from and to XML attachments.

Signature:

```
ora:processXSLTAttachment('template','input','href?','properties?')
```

Arguments:

- `template` - The XSLT template.
- `input` - The input data to be transformed.
- `href` - The location of the actual data.

- `properties` - The properties as defined in the `bpel.xml` file.

Property IDs:

- `namespace-uri: http://schemas.oracle.com/xpath/extension`
- `namespace-prefix: ora`

B.2.52 processXSQL

This function returns the result of the XSQL request.

Signature:

```
ora:processXSQL('template', 'input', 'properties')
```

Arguments:

- `template` - The XSLT template.
- `input` - The input data to be transformed.
- `properties` - The properties as defined in the `bpel.xml` file.

Property IDs:

- `namespace-uri: http://schemas.oracle.com/xpath/extension`
- `namespace-prefix: ora`

B.2.53 readBinaryFromFile

This function reads data from a file.

Signature:

```
ora:readBinaryFromFile(fileName)
```

Arguments:

- `fileName` - The file name from which to read data.

Property IDs:

- `namespace-uri: http://schemas.oracle.com/xpath/extension`
- `namespace-prefix: ora`

B.2.54 readFile

This function returns the content of the file.

Signature:

```
ora:readFile('fileName', 'nxsTemplate?', 'nxsRoot')
```

Arguments:

- `fileName` - The name of the file. This argument can also be an HTTP URL.

This function by default reads files relative to the suitcase JAR file for the process. If the file that you want to read is located in a different directory path, you must specify an extra directory slash (/) to indicate that this is an absolute path. For example:

```
ora:readFile('file:///c:/temp/test.doc')
```

If you specify only two directory slashes (//), you receive an error similar to the following:

```
XPath expression failed to execute.
Error while processing xpath expression,
the expression is "ora:readFile("file://c:/temp/test.doc")",
the reason is c. Verify the xpath query.
```

- `nxsdTemplate` - The NXSD template for the output
- `nxsdRoot` -The NXSD root

Property IDs:

- `namespace-uri`: `http://schemas.oracle.com/xpath/extension`
- `namespace-prefix`: `ora`

Note: Currently, the `readFile` function does not support the functionality to access files on a web server that requires authorization. If you tried to access such a file, then you get the following error:

```
java.io.IOException: Server returned HTTP response
code: 401 for URL
```

B.2.55 writeBinaryToFile

This function writes the binary bytes of a variable (or part of the variable) to a file of the given file name.

Signature:

```
ora:writeBinaryToFile(varName[, partName[, query]])
```

Arguments:

- `varName` - The name of the variable.
- `partName` - The name of the part in the `messageType` variable.
- `query` - The query string to a child of the root element.

Property IDs:

- `namespace-uri`: `http://schemas.oracle.com/xpath/extension`
- `namespace-prefix`: `ora`

B.2.56 BPEL Extension Functions

This section describes BPEL extension functions.

B.2.56.1 getLinkStatus

This function returns a boolean value indicating the status of the link. If the status of the link is positive the value is `true`, otherwise the value is `false`. This function can only be used in a `join` condition.

The `linkName` argument refers to the name of an incoming link for the activity associated with the join condition.

Signature:

```
bpws:getLinkStatus ('linkName')
```

Arguments:

- `variableName` - The source variable for the data.
- `propertyName` - The QName of the property.

Property IDs:

- `namespace-uri:`
`http://schemas.xmlsoap.org/ws/2003/03/business-process/`
- `namespace-prefix:` `bpws`

B.2.56.2 getVariableData

This function extracts arbitrary values from BPEL variables.

When only the first argument is present, the function extracts the value of the variable, which in this case must be defined using an XML schema simple type or element. Otherwise, the return value of this function is a node set containing the single node representing either an entire part of a message type (if the second argument is present and the third argument is absent) or the result of the selection based on the `locationPath` (if both optional arguments are present). If the given `locationPath` selects a node set of a size other than one during execution, the standard fault `bpws:selectionFailure` is thrown.

Signature:

```
bpws:getVariableData ('variableName', 'partName'?,  
'locationPath'?)
```

Arguments:

- `variableName` - The source variable for the data.
- `partName` - The part to select from the variable (optional).
- `locationPath` - Provides an absolute location path (with / meaning the root of the document fragment representing the entire part) to identify the root of a subtree within the document fragment representing the part (optional).

Property IDs:

- `namespace-uri:`
`http://schemas.xmlsoap.org/ws/2003/03/business-process/`
- `namespace-prefix:` `bpws`

B.2.56.3 getVariableProperty

This function extracts arbitrary values from BPEL variables.

If the given property selects a node set of a size other than one during execution, the standard fault `bpws:selectionFailure` is thrown.

Signature:

```
bpws:getVariableProperty ('variableName', 'propertyname')
```

Arguments:

- `variableName` - The source variable for the data.
- `propertyName` - The QName of the property.

- `locationPath` - Provides an absolute location path (with / meaning the root of the document fragment representing the entire part) to identify the root of a subtree within the document fragment representing the part (optional).

Property IDs:

- `namespace-uri:`
`http://schemas.xmlsoap.org/ws/2003/03/business-process/`
- `namespace-prefix:` `bpws`

B.2.57 Utility Functions

This section describes the utility functions.

B.2.57.1 `batchProcessActive`

This function returns the number of active processes in the batch.

Signature:

```
ora:batchProcessActive(String batchId, String processId)
```

Arguments:

- `batchId` - The ID of the batch.
- `processId` - The ID of the process.

Property IDs:

- `namespace-uri:``http://schemas.oracle.com/xpath/extension`
- `namespace-prefix:` `ora`

B.2.57.2 `batchProcessCompleted`

This function returns the number of completed processes in the batch.

Signature:

```
ora:batchProcessCompleted(String batchId, String processId)
```

Arguments:

- `batchId` - The ID of the batch.
- `processId` - The ID of the process.

Property IDs:

- `namespace-uri:``http://schemas.oracle.com/xpath/extension`
- `namespace-prefix:` `ora`

B.2.57.3 `format`

This function formats a message using Java's message format.

Signature:

```
ora:format(formatStrings, args+)
```

Arguments:

- `formatStrings` - The string of data to be formatted.

- `args+` - The arguments referenced by the format specifiers in the format string. If there are more arguments than format specifiers, the extra arguments are ignored. The number of arguments is variable and may be zero.

Property IDs:

- `namespace-uri`: `http://schemas.oracle.com/xpath/extension`
- `namespace-prefix`: `ora`

B.2.57.4 genEmptyElem

This function generates a list of empty elements for the given QName.

Signature:

```
ora:genEmptyElem('ElemQName', size?, 'TypeQName'?, xsiNil?)
```

Arguments:

- `ElemQName` - The first argument is the QName of the empty elements.
- `size` - The second optional integer argument for the number of empty elements. If missing, the default size is 1.
- `TypeQName` - The third optional argument is the QName, which is the `xsi:type` of the generated empty name. This `xsi:type` pattern matches `SOAPENC:Array`. If missing or an empty string, the `xsi:type` attribute is *not* generated.
- `xsiNil` - The fourth optional boolean argument is to specify whether the generated empty elements are `XSI - nil`, provided the element is XSD-nillable. The default is `false`. If missing or `false`, `xsi:nil` is *not* generated.

Property IDs:

- `namespace-uri`: `http://schemas.oracle.com/xpath/extension`
- `namespace-prefix`: `ora`

B.2.57.5 getChildElement

This function gets a child element for the given element.

Signature:

```
ora:getChildElement(element, index)
```

Arguments:

- `element` - The source for the data.
- `index` - The integer value of the child element index.

Property IDs:

- `namespace-uri`: `http://schemas.oracle.com/xpath/extension`
- `namespace-prefix`: `ora`

B.2.57.6 getMessage

This function gets a message based on the arguments.

Signature:

```
ora:getMessage(locale, relativeLocation, resourceName,  
resourceKey, resourceLocation?)
```

Arguments:

- `locale` - The locale of the message.
- `relativeLocation` - The subdirectory or message.
- `resourceName` - The name of the message resource.
- `resourceKey` - The key of the resource.
- `resourceLocation` - The location of the resource.

Property IDs:

- `namespace-uri: http://schemas.oracle.com/xpath/extension`
- `namespace-prefix: ora`

B.2.57.7 max-value-among-nodeset

This function returns the maximum value from a list of input numbers, the node-set `inputNumber`.

The node-set `inputNumber` can be a collection of text nodes or elements containing text nodes.

In the case of elements, the first text node's value is considered.

Signature:

```
oraext:max-value-among-nodeset(inputNumber as node-set)
```

Arguments:

- `inputNumber` - The node-set of input numbers.

Property IDs:

- `namespace-uri: http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.ExtFunc`
- `namespace-prefix: oraext`

B.2.57.8 min-value-among-nodeset

This function returns the minimum value from a list of input numbers, the node-set `inputNumbers`.

The node-set can be a collection of text nodes or elements containing text nodes.

In the case of elements, the first text node's value is considered.

Signature:

```
oraext:min-value-among-nodeset(inputNumbers as node-set)
```

Arguments:

- `inputNumber` - The node-set of input numbers.

Property IDs:

- `namespace-uri: http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.ExtFunc`
- `namespace-prefix: oraext`

B.2.57.9 square-root

This function returns the square root of `inputNumber`.

Example: `oraext:square-root(25)` returns 5

Signature:

```
oraext:square-root(inputNumber as number)
```

Arguments:

- `inputNumber` - The input number for which the function calculates the square root.

Property IDs:

- `namespace-uri:`
`http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.ExtFunc`
- `namespace-prefix:` `oraext`

B.2.57.10 translateFromNative

This function translates the input stream to an XML file.

Signature:

```
ora:translateFromNative('string', 'nxsdTemplate'?, 'nxsdRoot'?)
```

Arguments:

- `string` - The data to be converted into an XML file.
- `nxsdTemplate` - The XSD file used to define how the translation is performed.
- `nxsdRoot` - The root element defined in the XSD file.

Property IDs:

- `namespace-uri:` `http://schemas.oracle.com/xpath/extension`
- `namespace-prefix:` `ora`

B.2.57.11 translateToNative

Translates the XML to the native data.

Signature:

```
ora:translateFromNative('string', 'nxsdTemplate'?, 'nxsdRoot'?)
```

Arguments:

- `string` - The XML file to be converted into a string.
- `nxsdTemplate` - The XSD file used to define how the translation is performed.
- `nxsdRoot` - The root element defined in the XSD file.

Property IDs:

- `namespace-uri:` `http://schemas.oracle.com/xpath/extension`
- `namespace-prefix:` `ora`

B.2.57.12 translateFromNativeAttachment

This function translates the input stream to XML.

Signature:

```
ora:translateFromNativeAttachment('string', 'nxsdTemplate'?, 'nxsr
oot'?)
```

Arguments:

- `string` - The data to be converted into an XML file.
- `nxsdTemplate` - The XSD file used to define how the translation is performed.
- `nxsdRoot` - The root element defined in the XSD file.

Property IDs:

- `namespace-uri`: `http://schemas.oracle.com/xpath/extension`
- `namespace-prefix`: `ora`

B.2.57.13 translateToNativeAttachment

This function translates XML to the native data.

Signature:

```
ora:translateFromNativeAttachment('string', 'nxsdTemplate'?, 'nxsr
oot'?)
```

Arguments:

- `string` - The data to be converted into an XML file.
- `nxsdTemplate` - The XSD file used to define how the translation is performed.
- `nxsdRoot` - The root element defined in the XSD file.

Property IDs:

- `namespace-uri`: `http://schemas.oracle.com/xpath/extension`
- `namespace-prefix`: `ora`

B.3 Mediator XPath Extension Functions

This section describes the following functions:

B.3.1 GetComponentInstanceId

This function returns the component instance id.

Signature:

```
mdhr:getComponentInstanceId()
```

Arguments: There are no arguments for this function.

Property IDs:

- `namespace-uri`: `http://schemas.oracle.com/xpath/extension`
- `namespace-prefix`: `mdhr`

B.3.2 GetComponentName

This function returns the component name.

Signature:

`mdhr:getComponentName()`

Arguments: There are no arguments for this function.

Property IDs:

- `namespace-uri:` `http://schemas.oracle.com/xpath/extension`
- `namespace-prefix:` `mdhr`

B.3.3 `getCompositeInstanceId`

This function returns the composite instance id.

Signature:

`mdhr:getComponentInstanceId()`

Arguments: There are no arguments for this function.

Property IDs:

- `namespace-uri:` `http://schemas.oracle.com/xpath/extension`
- `namespace-prefix:` `mdhr`

B.3.4 `getCompositeName`

This function returns the composite name.

Signature:

`mdhr:getCompositeName()`

Arguments: There are no arguments for this function.

Property IDs:

- `namespace-uri:` `http://schemas.oracle.com/xpath/extension`
- `namespace-prefix:` `mdhr`

B.3.5 `getHeader`

This function returns the value of an XPath expression from the mediator message header.

Note: The `getHeader` function works only when both parameters are specified.

Signature:

`mdhr:getHeader(xpath as string, namespaces as string)`

Arguments:

- `xpath:` Refers to the path you traverse from the schema.
- `namespaces:` Refers to the abstract container that contains the context of the XPath expression. This argument is not optional. Namespace declarations are in the following form:

`'prefix=namespace;`

Note the semicolon after the namespace declaration. For example:

```
getHeader("in.header.ns9_name/ns9:name/ns9:first", "ns9=http://example.com;")
```

In the XSLT Mapper in Oracle JDeveloper, drag the `getHeader` function into the mapper. In the Edit Function - `getHeader` dialog, click **Add**. The `namespace` argument is added for you to enter the required information.

Property IDs:

- `namespace-uri: http://schemas.oracle.com/xpath/extension`
- `namespace-prefix:mdhr`

B.3.6 getECID

This function returns the ECID.

Signature:

```
mdhr:getECID()
```

Arguments: There are no arguments for this function.

Property IDs:

- `namespace-uri: http://schemas.oracle.com/xpath/extension`
- `namespace-prefix: mdhr`

B.3.7 getParentComponentInstanceId

This function returns the mediator instance parent component instance id.

Signature:

```
mdhr:getParentComponentInstanceId()
```

Arguments: There are no arguments for this function.

Property IDs:

- `namespace-uri: http://schemas.oracle.com/xpath/extension`
- `namespace-prefix: mdhr`

B.3.8 setCompositeInstanceTitle

This function sets a title to the composite instance that can be later used as one of the criteria in searching the instances. This function returns the same string that is passed as the argument.

Signature:

```
mdhr:setCompositeInstanceTitle(title)
```

Arguments:

- `title` - Specifies the composite instance title. This can be specified as an XPath expression on the message payload.

Property IDs:

- `namespace-uri: http://schemas.oracle.com/xpath/extension`
- `namespace-prefix: mdhr`

B.4 Advanced Functions

This section describes the advanced functions.

B.4.1 create-nodeset-from-delimited-string

The function takes a delimited string and returns a nodeSet.

Signature:

```
oraext:create-nodeset-from-delimited-string(qname,  
delimited-string, delimiter)
```

Arguments:

- `qname` - The qualified name in which each node in the node set must be created. The QName can be represented in two forms:
 - `task:assignee`
 - `{http://mytask/task}assignee`
- `delimited-string` - The sting of elements separated by the delimiter.
- `delimiter` - The character that separates the items in the input string; for example, a comma or a semicolon.

Property IDs:

- `namespace-uri`:
`http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.ExtFunc`
- `namespace-prefix`: `oraext`

B.4.2 generate-guid

The function generates a unique GUID.

Signature:

```
oraext:generate-guid()
```

Arguments: There are no arguments for this function.

Property IDs:

- `namespace-uri`:
`http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.ExtFunc`
- `namespace-prefix`: `oraext`

B.4.3 lookupPopulatedColumns

This function is used to look up a cross-reference column for a single value or multiple values corresponding to a value in a reference column.

Signature:

```
xref:lookupPopulatedColumns(tableName, columnName, value, needAnException)
```

Arguments:

- `xrefTableName`: The name of the reference table.

- `xrefColumnName`: The name of the reference column.
- `xrefValue`: The value corresponding to reference column name.
- `needAnException`: If this value is set to true, then an exception is thrown when no value is found in the referenced column. Otherwise, an empty node-set is returned.

Property IDs:

- `namespace-uri`: `http://www.oracle.com/XSL/Transform/java/oracle.tip.xref.xpath.XRefXPathFunctions`
- `namespace-prefix`: `xref`

B.4.4 lookupValue

The function returns a string by looking up the value for the target column in a domain value map, where the source column contains the given source value.

Signature:

```
dvm:lookupValue(dvmLocation, sourceColumnName, sourceValue, targetColumnName, defaultValue)
```

Arguments:

- `dvmLocation`: The domain value map URI.
- `sourceColumnName`: The source column name.
- `sourceValue`: The source value (an XPath expression bound to the source document of the XSLT transformation).
- `targetColumnName`: The target column name.
- `defaultValue`: If the value is not found, then the default value is returned.
- `QualifierSourceColumn`: The name of the qualifier column.
- `QualifierSourceValue`: The value of the qualifier.

Property IDs:

- `namespace-uri`:
`http://www.oracle.com/XSL/Transform/java/oracle.tip.dvm.LookupValue`
- `namespace-prefix`: `dvm`

B.4.5 lookupValue1M

The function returns an XML document fragment containing values for multiple target columns of a domain value map, where the value for source column equals the source value.

Signature:

```
dvm:lookupValue1M(dvmLocation, sourceColumnName, sourceValue, targetColumnName1, targetColumnName2...)
```

Arguments:

- `dvmMetadataURI` - The domain value map URI.
- `SourceColumnName` - The source column name.

- `SourceValue` - The source value (an XPath expression bound to the source document of the XSLT transformation).
- `TargetColumnName` - The name of the target columns. At least one column name should be specified. The question mark symbol (?) indicates that you can specify multiple target column names.

Property IDs:

- `namespace-uri`:
`http://www.oracle.com/XSL/Transform/java/oracle.tip.dvm.LookupValue`
- `namespace-prefix`: `dvm`

B.4.6 lookupXRef

This function is used to look up a cross-reference column for a value that corresponds to a value in a reference column.

Signature:

```
xref:lookupXRef (tableName, referenceColumnName, referenceValue, columnName, needAnException)
```

Arguments:

- `xrefLocation`: The cross-reference URI.
- `xrefReferenceColumnName`: The name of the reference column.
- `xrefReferenceValue`: The value corresponding to reference column name.
- `xrefColumnName`: The name of the column to be looked up for the value.
- `needAnException`: : When value is set to true, an exception is thrown if the value is not found, else an empty value is returned.

Property IDs:

- `namespace-uri`:`http://www.oracle.com/XSL/Transform/java/oracle.tip.xref.xpath.XRefXPathFunctions`
- `namespace-prefix`: `xref`

B.4.7 lookupXRef1M

This function is used to look up a cross-reference column for multiple values corresponding to a value in a reference column.

Signature:

```
xref:lookupXRef1M (tableName, referenceColumnName, referenceValue, columnName, needAnException)
```

Arguments:

- `xrefLocation`: The cross-reference URI.
- `xrefReferenceColumnName`: The name of the reference column.
- `xrefReferenceValue`: The value corresponding to reference column name.
- `xrefColumnName`: The name of the column to be looked up for the value.
- `needAnException`: If this value is set to true, then an exception is thrown when the referenced value is not found. Else, an empty node-set is returned.

Property IDs:

- namespace-uri: `http://www.oracle.com/XSL/Transform/java/oracle.tip.xref.xpath.XRefXPathFunctions`
- namespace-prefix: `xref`

B.4.8 lookup-xml

This function returns the string value of an element defined by `lookupXPath` in an XML file (`docURL`) given its parent XPath (`parentXPath`), the key XPath (`keyXPath`), and the value of the key (`key`).

Example: `oraext:lookup-xml('file:/d:/country_data.xml', '/Countries/Country', 'Abbreviation', 'FullName', 'UK')` returns the value of the element `FullName` child of `/Countries/Country` where `Abbreviation = 'UK'` is in the file `D:\country_data.xml`.

Signature:

```
oraext:lookup-xml(docURL, parentXPath, keyXPath, lookupXPath, key)
```

Arguments:

- `docURL` - The XML file
- `parentXPath` - The parent XPath
- `keyXPath` - The key XPath
- `lookupXPath` - The lookup XPath
- `key` - The key value

Property IDs:

- namespace-uri: `http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.ExtFunc`
- namespace-prefix: `oraext`

B.4.9 markForDelete

The function is used to delete a value in a cross-reference table. The value in the column is marked as deleted. This function returns true if deletion is successful else returns false.

Signature:

```
xref:markForDelete(tableName, columnName, value)
```

Arguments:

- `xrefTableName`: The cross-reference table name.
- `xrefColumnName`: The name of the column from which you want to delete a value.
- `xrefValueToDelete`: The value to be deleted.

Property IDs:

- namespace-uri: `http://www.oracle.com/XSL/Transform/java/oracle.tip.xref.xpath.XRefXPathFunctions`

- namespace-prefix: xref

B.4.10 populateXRefRow

The function populates the column name in the cross-reference table (XREF) where the reference column has the reference value.

Signature:

```
xref:populateXRefRow(tableName, referenceColumnName, referenceValue, columnName, value, mode)
```

Arguments:

- xrefLocation: The cross-reference URI.
- xrefReferenceColumnName: The name of the reference column.
- xrefReferenceValue: The value corresponding to reference column name.
- xrefColumnName: The name of the column to be looked up for the value.
- xrefvalue: The value corresponding to reference column name.
- xrefmode: The name of the XREF population mode.

Property IDs:

- namespace-uri: <http://www.oracle.com/XSL/Transform/java/oracle.tip.xref.xpath.XRefXPathFunctions>
- namespace-prefix: xref

B.4.11 populateXRefRow1M

The function populates the column multiple values in the cross-reference table (XREF) where the reference column has the reference value.

Signature:

```
xref:populateXRefRow1M(tableName, referenceColumnName, referenceValue, columnName, value, mode)
```

Arguments:

- xrefLocation: The cross-reference URI.
- xrefReferenceColumnName: The name of the reference column.
- xrefReferenceValue: The value corresponding to reference column name.
- xrefColumnName: The name of the column to be looked up for the value.
- xrefvalue: The value corresponding to reference column name.
- xrefmode: The name of the XREF population mode.

Property IDs:

- namespace-uri: <http://www.oracle.com/XSL/Transform/java/oracle.tip.xref.xpath.XRefXPathFunctions>
- namespace-prefix: xref

B.5 Workflow Service Functions

This section describes the workflow service functions.

B.5.1 clearTaskAssignees

This function clears the current task assignees.

Signature:

```
hwf:clearTaskAssignees(taskID)
```

Arguments:

- `task` - The task ID of the task.

Property IDs:

- `namespace-uri`: `http://xmlns.oracle.com/bpel/workflow/xpath`
- `namespace-prefix`: `hwf`

B.5.2 createWordMLDocument

This function creates a Microsoft Word ML document as a base 64-encoded string.

Signature:

```
hwf:createWordMLDocument(node, xsltURI)
```

Arguments:

- `node` - The node is an XML Node that is an input to the transformation.
- `xsltURI` - The XSLT used to transform the node (the first argument) to Microsoft Word ML.

Property IDs:

- `namespace-uri`: `http://xmlns.oracle.com/bpel/workflow/xpath`
- `namespace-prefix`: `hwf`

B.5.3 getNotificationProperty

This function retrieves a notification property. The function evaluates to corresponding values for each notification. Only use this function in the notification content XPath expression. If used elsewhere, it returns `null`.

Signature:

```
hwf:getNotificationProperty(propertyName)
```

Arguments:

- `propertyName` - The name of the notification property. It can be one of the following values:
 - `recipient` - The recipient of the notification.
 - `recipientDisplay` - The display name of the recipient.
 - `taskAssignees` - The task assignees.
 - `taskAssigneesDisplay` - The display names of the task assignees.
 - `locale` - The locale of the recipient.
 - `taskId` - The task ID of the task for which the notification is meant.
 - `taskNumber` - The task number of the task for which the notification is meant.

- `appLink` - The HTML link to the Oracle BPM Worklist task details page.

Property IDs:

- `namespace-uri`: `http://xmlns.oracle.com/bpel/workflow/xpath`
- `namespace-prefix`: `hwf`

B.5.4 `getNumberOfTaskApprovals`

This function computes the number of times the task was approved.

Signature:

```
hwf:getNumberOfTaskApprovals(taskId)
```

Arguments:

- `taskId` - The ID of the task

Property IDs:

- `namespace-uri`: `http://xmlns.oracle.com/bpel/workflow/xpath`
- `namespace-prefix`: `hwf`

B.5.5 `getPreviousTaskApprover`

This function retrieves the previous task approver.

Signature:

```
hwf:getPreviousTaskApprover(taskId)
```

Arguments:

- `taskId` - The ID of the task

Property IDs:

- `namespace-uri`: `http://xmlns.oracle.com/bpel/workflow/xpath`
- `namespace-prefix`: `hwf`

B.5.6 `getTaskAttachmentByIndex`

This function retrieves the task attachment at the specified index.

Signature:

```
hwf:getTaskAttachmentByIndex(taskId, attachmentIndex)
```

Arguments:

- `taskId` - The task ID of the task.
- `attachmentIndex` - The index of the attachment. The index begins from 1. The `attachmentIndex` argument can be a node whose value evaluates to the index number as a string (all node values are strings). If specified statically, it can be specified as `'1'`.

Property IDs:

- `namespace-uri`: `http://xmlns.oracle.com/bpel/workflow/xpath`
- `namespace-prefix`: `hwf`

B.5.7 `getTaskAttachmentByName`

This function retrieves the task attachment by the attachment name.

Signature:

```
hwf:getTaskAttachmentByName(taskId, attachmentName)
```

Arguments:

- `taskId` - The task ID of the task.
- `attachmentName` - The name of the attachment.

Property IDs:

- `namespace-uri:http://xmlns.oracle.com/bpel/workflow/xpath`
- `namespace-prefix:hwf`

B.5.8 `getTaskAttachmentContents`

This function retrieves the task attachment contents by the attachment name.

Signature:

```
hwf:getTaskAttachmentContents(taskId, attachmentName)
```

Arguments:

- `taskId` - The task ID of the task.
- `attachmentName` - The name of the attachment.

Property IDs:

- `namespace-uri:http://xmlns.oracle.com/bpel/workflow/xpath`
- `namespace-prefix:hwf`

B.5.9 `getTaskAttachmentsCount`

This function retrieves the number of task attachments.

Signature:

```
hwf:getTaskAttachmentsCount(taskId)
```

Arguments:

- `taskId` - The task ID of the task.

Property IDs:

- `namespace-uri:http://xmlns.oracle.com/bpel/workflow/xpath`
- `namespace-prefix:hwf`

B.5.10 `getTaskResourceBundleString`

This function returns the internationalized resource value from the resource bundle associated with a task definition.

Signature:

```
hwf:getTaskResourceBundleString(taskId, key, locale?)
```

Arguments:

- `taskId` - The task ID of the task.
- `key` - The key to the resource.
- `locale` - (Optional) The locale. This value defaults to system locale. This returns a `resourceString` XML element in the namespace `http://xmlns.oracle.com/bpel/services/taskService`, which contains the string from the resource bundle.

Property IDs:

- `namespace-uri`: `http://xmlns.oracle.com/bpel/workflow/xpath`
- `namespace-prefix`: `hwf`

B.5.11 wfDynamicGroupAssign

This function gets the name of an identity service group, selected according to the specified assignment pattern. The group is selected from either the subordinate groups of the specified group (if a single group name is supplied), or from the list of groups (if a list of user names is supplied). If the identity service is configured with multiple realms, the realm name for the group and groups must also be supplied. Additional assignment pattern specific parameters can be supplied. These additional parameters are optional, depending on the details of the specific assignment pattern used.

There are two signatures of this function.

Signature 1:

```
hwf:wfDynamicGroupAssign('patternName', 'groupName', 'realmName'?,
'patternParam1'?, 'patternParam2'?, ..., 'patternParamN'?)
```

Argument 1:

- `patternName` - The name of the assignment pattern (for example, `ROUND_ROBIN`).
- `groupName` - The name of the group from which to select a subordinate group.
- `realmName` - The name of the identity service realm to which the group belongs.
- `patternParam1...patternParamN` - Any additional parameters required by the assignment pattern implementation (may be optional, depending on pattern).

Signature 2:

```
hwf:wfDynamicGroupAssign('patternName', 'groupList', 'realmName'?,
'patternParam1'?, 'patternParam2'?, ..., 'patternParamN'?)
```

Argument 2:

- `patternName` - The name of the assignment pattern (for example, `ROUND_ROBIN`).
- `groupList` - The list of groups from which to select a group.
- `realmName` - The name of the identity service realm to which the groups belong.
- `patternParam1...patternParamN` - Any additional parameters required by the assignment pattern implementation (may be optional, depending on the pattern).

Property IDs:

- `namespace-uri`: `http://xmlns.oracle.com/bpel/workflow/xpath`
- `namespace-prefix`: `hwf`

B.5.12 wfDynamicUserAssign

This function returns the name of an identity service user, selected according to the specified assignment pattern. The user is selected from either the subordinate users of the specified group (if a single group name is supplied), or from the list of users (if a list of user names is supplied). If the identity service is configured with multiple realms, the realm name for the group and users must also be supplied. Additional assignment pattern specific parameters can be supplied. These additional parameters are optional, depending on the details of the specific assignment pattern used.

There are two signatures for this function.

Signature 1:

```
hwf:wfDynamicUserAssign('patternName', 'groupName', 'realmName'?, '
patternParam1'?, . . . . , 'patternParam2'?, . . . . , 'patternParamN'?)
```

Arguments 1:

- `patternName` - The name of the assignment pattern (for example, ROUND_ROBIN).
- `groupName` - The name of the group from which to select a subordinate user.
- `realmName` - The name of the identity service realm to which the group belongs.
- `patternParam1` ... `patternParamN` - Any additional parameters required by the assignment pattern implementation (may be optional, depending on the pattern).

Signature 2:

```
hwf:wfDynamicUserAssign(patternName, userList, realmName?, patternP
aram1?, patternParam2?, . . . . , patternParamN?)
```

Arguments 2:

- `patternName` - The name of the assignment pattern (for example, ROUND_ROBIN).
- `userList` - The list of users from which to select a user.
- `realmName` - The name of the identity service realm to which the users belong.
- `patternParam1` . . . `patternParamN` - Any additional parameters required by the assignment pattern implementation (may be optional, depending on the pattern).

Property IDs:

- `namespace-uri`: `http://xmlns.oracle.com/bpel/workflow/xpath`
- `namespace-prefix`: `hwf`

B.5.13 Identity Service Functions

This section describes the identity service functions.

B.5.13.1 getDefaultRealmName

This function returns the default realm name.

Signature:

```
ids:getDefaultRealmName()
```

Arguments: There are no arguments for this function.

Property IDs:

- namespace-uri:
http://xmlns.oracle.com/bpel/services/IdentityService/xpath
- namespace-prefix: ids

B.5.13.2 getGroupProperty

This function returns the property value for the given group. If the group or attribute does not exist, it returns null.

Signature:

```
ids:getGroupProperty(groupName, attributeName, realmName)
```

Arguments:

- groupName - String or element containing the group whose attribute must be retrieved.
- attributeName - String or element containing the name of the group attribute. The name is one of the following values:
 1. displayName
 2. emailIf the identity service uses the LDAP providerType or JAZN LDAP-based providers, configure the LDAP server to enable searching by those attributes.
- realmName - The realm name. This is optional. If not specified, the default realm is assumed.

Property IDs:

- namespace-uri:
http://xmlns.oracle.com/bpel/services/IdentityService/xpath
- namespace-prefix: ids

B.5.13.3 getManager

This function gets the manager of a given user. If the user does not exist or there is no manager for this user, it returns null.

Signature:

```
ids:getManager(userName, realmName)
```

Arguments:

- userName - The user name.
- realmName - The realm name. This is optional. If not specified, the default realm is assumed.

Property IDs:

- namespace-uri: http://xmlns.oracle.com/bpel/services/IdentityService/xpath
- namespace-prefix: ids

B.5.13.4 getReportees

This function gets the reportees of the user. If the user does not exist, it returns null. The function returns a list of nodes. Each node in the list is called user.

Signature:

```
ids:getReportees(userName, upToLevel, realmName)
```

Arguments:

- `userName` - The user name.
- `upToLevel` - Defines the levels of indirect reportees to be included into the result. If the value is 1, it returns only direct reportees. If the value is -1, it returns all levels of reportees. It can be either an element with value `xsd:number` or a string, for example '1'.
- `realmName` - The realm name. This is optional and if not specified, the default realm is assumed.

Property IDs:

- `namespace-uri:http://xmlns.oracle.com/bpel/services/IdentityService/xpath`
- `namespace-prefix:ids`

B.5.13.5 getSupportedRealmNames

This function returns the supported realm names.

Signature:

```
ids:getSupportedRealms()
```

Property IDs:

- `namespace-uri:http://xmlns.oracle.com/bpel/services/IdentityService/xpath`
- `namespace-prefix:ids`

B.5.13.6 getUserProperty

This function returns the property of the user. If the user or attribute does not exist, it returns null.

Signature:

```
ids:getUserProperty(userName, attributeName, realmName)
```

Arguments:

- `userName` - String or element containing the user whose attribute must be retrieved.
- `attributeName` - The name of the user attribute. The attribute name is one of the following values:
 1. `givenName`
 2. `middleName`
 3. `sn`
 4. `displayName`
 5. `mail`
 6. `telephoneNumber`
 7. `homephone`

8. `mobile`
9. `facsimile`
10. `pager`
11. `preferredlanguage`
12. `title`
13. `manager`

If the identity service uses the LDAP `providerType` or JAZN LDAP-based providers, configure the LDAP server to enable searching by those attributes.

- `realmName` - The realm name. This is optional. If not specified, the default realm name is assumed.

Property IDs:

- `namespace-uri`:
`http://xmlns.oracle.com/bpel/services/IdentityService/xpath`
- `namespace-prefix`: `ids`

B.5.13.7 `getUserRoles`

This function gets the user roles. This function returns a list of objects, either application roles or groups, depending on the `roleType`. If the user or role does not exist, it returns `null`.

Signature:

```
ids:getUserRoles(userName, roleType, direct)
```

Arguments:

- `userName` - String or element containing the user whose roles are to be retrieved.
- `roleType` - The role type that takes one of three values: `ApplicationRole`, `EnterpriseRole`, or `AnyRole`.
- `direct` - A string or element indicating if direct or indirect roles must be fetched. This is optional. If not specified, only direct roles are fetched. This is either `xsd:boolean` or string `true/false`.

Property IDs:

- `namespace-uri`:`http://xmlns.oracle.com/bpel/services/IdentityService`
- `namespace-prefix`: `ids`

B.5.13.8 `getUsersInGroup`

This function gets the users in a group. If the group does not exist, it returns `null`. The function returns a list of nodes. Each node in the list is called `user`.

Signature:

```
ids:getUsersInGroup(groupName, direct, realmName)
```

Arguments:

- `groupName` - The group name.

- `direct` - A boolean flag. If `true`, the function returns direct user grantees; otherwise, all user grantees are returned. It can be either an element with value `xsd:boolean` or string `'true'/'false'`.
- `realmName` - The realm name. This is optional. If not specified, the default realm name is assumed.

Property IDs:

- `namespace-uri:`
`http://xmlns.oracle.com/bpel/services/IdentityService/xpath`
- `namespace-prefix:` `ids`

B.5.13.9 isUserInRole

This function verifies if a user has a given role.

Signature:

```
ids:isUserInRole(userID, roleName, realmName)
```

Arguments:

- `userID` - A string or element containing the user whose participation in the role must be verified.
- `roleName` - The role name.
- `realmName` - The realm name. This is optional. If not specified, the default realm name is assumed.

Property IDs:

- `namespace-uri:``http://xmlns.oracle.com/bpel/services/IdentityService/xpath`
- `namespace-prefix:` `ids`

B.5.13.10 lookupGroup

This function gets the group. If the group does not exist, it returns null.

Signature:

```
ids:lookupGroup(groupName, realmName)
```

Arguments:

- `groupName` - The group name.
- `realmName` - The realm name. This is optional. If not specified, the default realm name is assumed.

Property IDs:

- `namespace-uri:``http://xmlns.oracle.com/bpel/services/IdentityService/xpath`
- `namespace-prefix:` `ids`

B.5.13.11 lookupUser

This function gets the user object. If the user does not exist, it returns null.

Signature:

```
ids:lookupUser(userName, realmName)
```

Arguments:

- `userName` - The user name.
- `realmName` - The realm name. This is optional. If not specified, the default realm name is assumed.

Property IDs:

- `namespace-uri:`
`http://xmlns.oracle.com/bpel/services/IdentityService/xpath`
- `namespace-prefix:` `ids`

B.6 Using the XPath Building Assistant

You can use the XPath Building Assistant to create XPath expressions.

B.6.1 XPath Building Assistant Description

Several dialogs enable you to specify XPath expressions at several places, including:

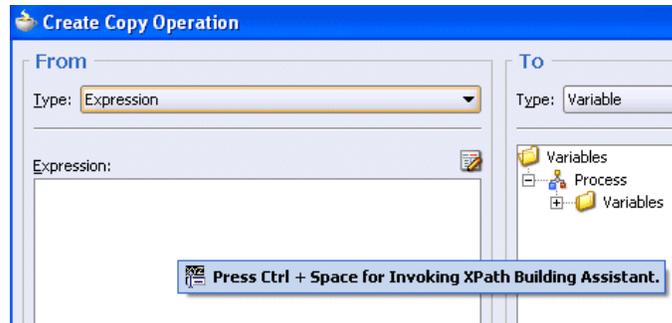
- **Expression** field of the XPath Expression Builder
- **Expression** field of an operation created under the **Copy Operation** tab of assign activities
- **Expression** field of the while, wait, switch, and pick (onAlarm branch) activities
- Edit XPath Expression and Edit Function dialogs of the XSLT Mapper

Manually specifying long and complex expressions is supported, but can be a cumbersome and error-prone process. The XPath Building Assistant provides the following set of features that simplify this process:

- Automatic completion of the following:
 - Elements and attributes
 - Functions
 - BPEL variables and parts
- Function parameter tool tips
- Syntactic and semantic validation of XPath expressions

B.6.2 Starting the XPath Building Assistant

Start the XPath Building Assistant by clicking inside the **Expression** field and pressing **Ctrl** and then the **space bar**. The XPath Building Assistant is available within all fields of the Oracle JDeveloper and XSLT Mapper function dialogs that require XPath expressions.



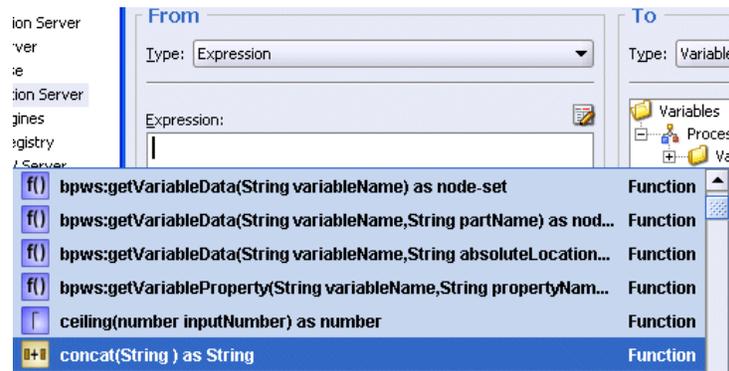
B.6.3 Using the XPath Building Assistant in Oracle JDeveloper: Step-By-Step Example

This section provides an example of using the XPath Building Assistant to build an expression in the **From** section of the **Expression** field of the Create Copy Operation dialog. This example models an XPath Expression that appends a string value to **OrderComments** within a purchase order. The purchase order element is part of one of the available BPEL variables.

1. Place the cursor inside the **Expression** field.



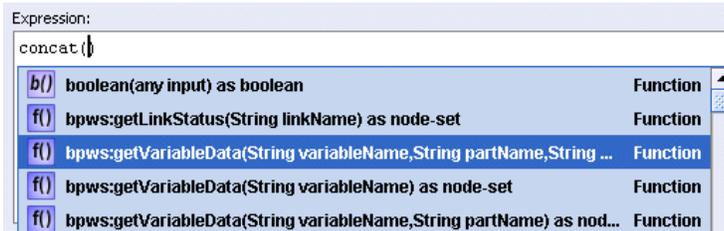
2. Press **Ctrl** and then the **space bar** to display a list of values for building an expression.



3. Make a selection from the list (for this example, **concat(String) as String**) in either of the following ways:
 - Scroll down the list and double-click **concat(String) as String**.
 - Enter the letter **c** to display only items starting with that letter and double-click **concat(String) as String**.

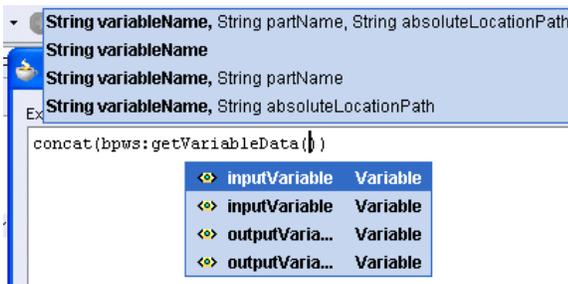
This value is added to the **Expression** field. The list automatically displays again with different options and prompts you to enter the next portion of the XPath expression.

4. Select and double-click the next portion (for this example, the second entry for **bpws**):

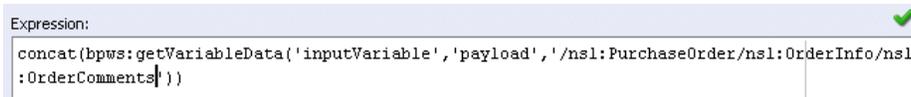


This value is added to the **Expression** field. The list automatically displays again and prompts you to enter the next portion of the XPath expression.

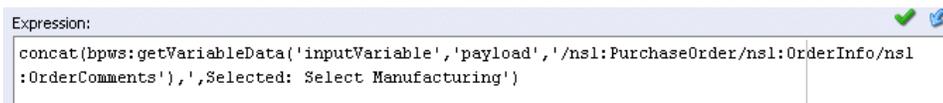
5. Select and double-click **inputVariable**.



6. Continue this process to build the remaining parts of the XPath expression (for this example, double-click **payload > ns1:/PurchaseOrder > ns1:/OrderInfo > ns1:OrderComments** as they appear).



7. Manually add text as appropriate (for this example, **','Selected: Select Manufacturing'**). If needed, you can also manually enter logical operators (such as **>**, **<**, and so on).



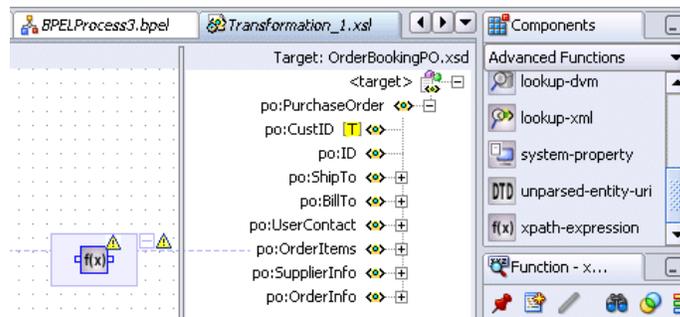
Note: Instead of using double-clicks on the XPath Building Assistant popups, you can also use the **Enter** key to make the selection. If your expression is complete, but you are still being prompted to enter information, press **Esc**. This closes the list.

B.6.4 Using the XPath Building Assistant in the XSLT Mapper

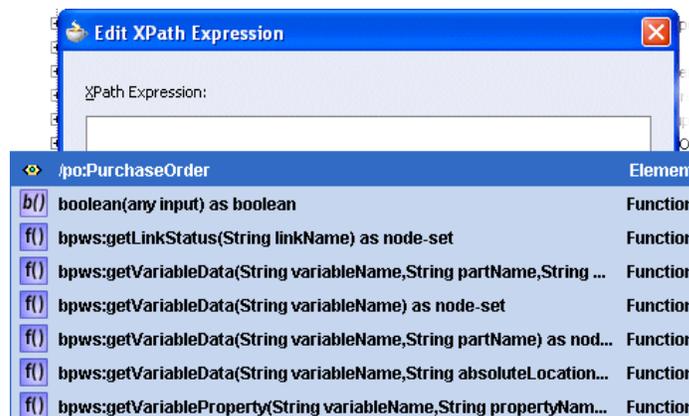
This section provides an example of using the XPath Building Assistant to build an expression in the Edit XPath Expression dialog of the XSLT Mapper.

1. Go to the transformation dialog.
2. Select **Advanced Functions** from the **Component Palette** list.
3. Scroll down the list to the **xpath-expression**.

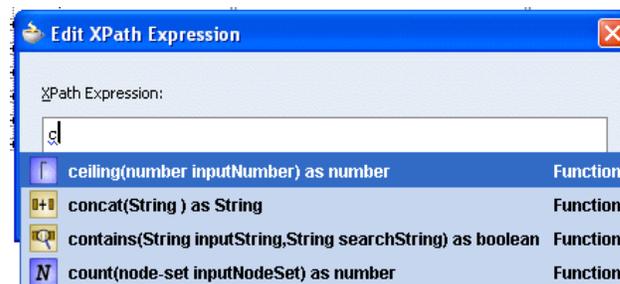
4. Drag and drop the **xpath-expression** into the transformation dialog.



5. Double-click the function to display the Edit XPath Expression dialog.
6. Click the cursor inside the **XPath Expression** field.
7. Press **Ctrl** and then the **space bar** to display a list of values for building an expression.



8. Make a selection from the list (for this example, **concat(String) as String**) in either of the following ways:
 - Scroll down the list and double-click **concat(String) as String**.
 - Enter the letter **c** to display only items starting with that letter and double-click **concat(String) as String**.



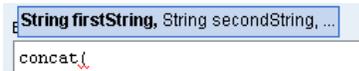
This selection is added to the **XPath Expression** field. The list automatically displays again with different options and prompts you to enter the next portion of the XPath expression.

9. Continue this process to build the remaining parts of the XPath expression (for this example, double-click **po:PurchaseOrder** > **po:ShipTo** > **po:Name** > **po:First** as they appear).

10. Continue this process to build the remaining parts of the expression.
11. Click **OK** to close the Edit XPath Expression dialog when complete.

B.6.5 Function Parameter Tool Tips

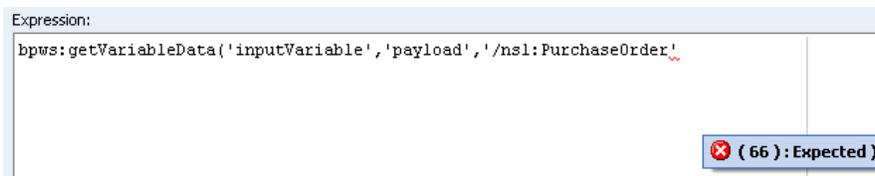
Function parameter tool tips display the expected arguments of a chosen XPath function. For example, if you manually enter the function `concat`, and then enter `(`, the parameter tool tip appears and displays the expected arguments of the `concat` function. The current argument name of the function is highlighted in bold.



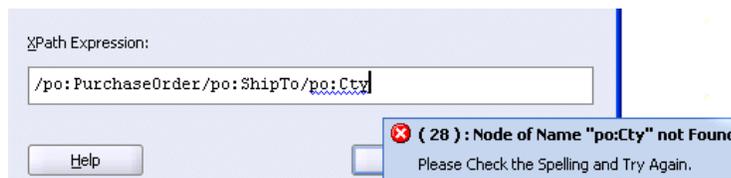
Once you finish specifying one argument, and enter a comma to move to the next argument, the tool tip updates itself to highlight the second argument name in bold, and so on. While editing existing XPath expressions that contain functions, you can re-invoke parameter tool tips by positioning the cursor within the function and then pressing a combination of the **Ctrl**, **Shift**, and **space bar** keys.

B.6.6 Syntactic and Semantic Validation

Within Oracle JDeveloper, an XPath expression is considered syntactically valid if it conforms to the XPath 1.0 specification. The XPath Building Assistant warns you about syntactically incorrect XPath expressions (for example, a missing parenthesis or apostrophe) by underlining the erroneous area in red. Drag the mouse pointer over this area. The error message displays as a tool tip. The red underlining error disappears after you make corrections.



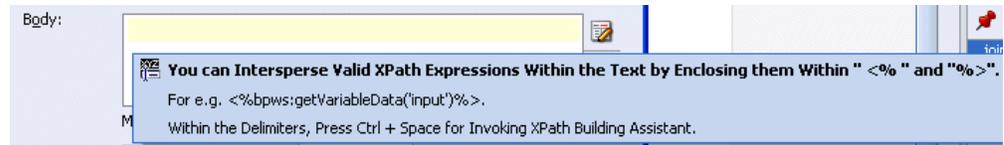
Syntactically valid XPath expressions may be semantically invalid. This can cause unexpected errors at runtime. For example, you can misspell the name of an element, variable, function, or part. The XPath Building Assistant warns you about semantic errors by underlining the erroneous area in blue. Drag the mouse pointer over this area. The error message displays as a tool tip. The blue underlining error disappears after you make corrections.



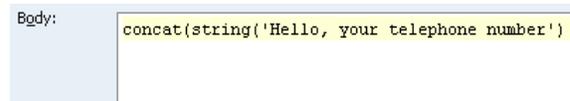
B.6.7 Creating Expressions with Free Form Text and XPath Expressions

You can mix free form text with XPath expressions in some dialogs.

1. Place your cursor over the field to display a popup message that describes this functionality.



2. Enter free form text (in this example, 'Hello, your telephone number').

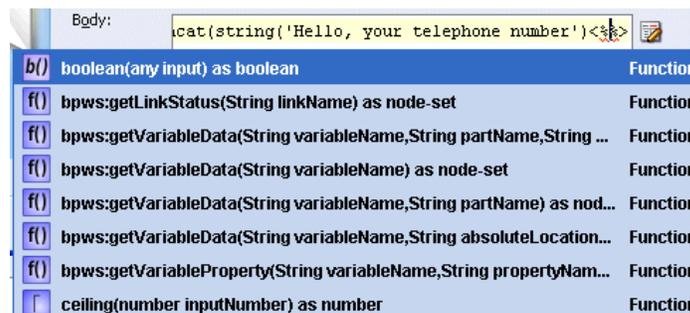


3. Enter `$\%$` when you are ready to invoke the XPath Building Assistant.

```
concat(string('Hello, your telephone number'))<math>\%</math>
```

A red underline appears, which indicates that you are being prompted to add information.

4. Press **Ctrl** and then the **space bar** to invoke the XPath Building Assistant.



5. Scroll down the list and double-click the value you want.
6. Continue this process to build the remaining parts of the expression.

B.7 Creating User-Defined XPath Extension Functions

You can create user-defined (custom) XPath extension functions for use in Oracle SOA Suite. These functions can be created for the following components:

- Oracle BPEL Process Manager
- Oracle Mediator
- XSLT Mapper
- Human workflow
- Shared by all of these components

XPath extension functions in Oracle SOA Suite adhere to the following standards:

- A single schema defines the configuration syntax for both system functions and user-defined functions.
- XPath functions are categorized based on usage (Oracle BPEL Process Manager, Oracle Mediator, human workflow, XSLT Mapper, and those commonly used by all).
- System functions are separated from user-defined functions.

- A repository hosts both system function configuration files and user-defined function configuration files.
- A repository hosts user-defined function implementation JAR files and automatically makes them available for the Java Virtual Machine (JVM) (class loaders).

As a best practice, follow these conventions for creating functions:

- If possible, write functions that can be shared across all components. Functions shared by all components can be created in a configuration file named `ext-soa-xpath-functions-config.xml`. Note that you must implement XSLT Mapper functions differently than functions for Oracle BPEL Process Manager, Oracle Mediator, and human workflow.

For more information about description of these implementation differences, see [Section B.7.1, "How to Implement User-Defined XPath Extension Functions"](#).

- If you create a function for one component that cannot be used by others (for example, a function for Oracle BPEL Process Manager that cannot be used by Oracle Mediator or human workflow), then create that function in the configuration file specific to that component. For this example, the Oracle BPEL Process Manager function must be created in a configuration file named `ext-bpel-xpath-functions-config.xml`.

[Example B-1](#) shows the function schema used by system and user-defined functions.

Example B-1 Function Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:tns="http://xmlns.oracle.com/soa/config/xpath"
  targetNamespace="http://xmlns.oracle.com/soa/config/xpath"
  elementFormDefault="qualified">
  <element name="soa-xpath-functions" type="tns:XpathFunctionsConfig"/>
  <element name="function" type="tns:XpathFunction"/>
  <complexType name="XpathFunctionsConfig">
    <sequence>
      <element ref="tns:function" minOccurs="1" maxOccurs="unbounded"/>
    </sequence>
    <attribute name="resourceBundle" type="string"/>
    <attribute name="version" type="string"/>
  </complexType>

  <complexType name="XpathFunction">
    <sequence>
      <element name="className" type="string"/>
      <element name="return">
        <complexType>
          <attribute name="type" type="tns:XpathType"
            use="required"/>
        </complexType>
      </element>
      <element name="params" type="tns:Params" minOccurs="0"
        maxOccurs="1"/>
      <element name="desc">
        <complexType>
          <simpleContent>
            <extension base="string">
              <attribute name="resourceKey"
                type="string"/>
            </extension>
          </simpleContent>
        </complexType>
      </element>
    </sequence>
  </complexType>
</schema>
```

```

        </extension>
    </simpleContent>
</complexType>
</element>
<element name="detail" minOccurs="0">
    <complexType>
        <simpleContent>
            <extension base="string">
                <attribute name="resourceKey"
                    type="string"/>
            </extension>
        </simpleContent>
    </complexType>
</element>
<element name="icon" minOccurs="0">
<complexType>
    <simpleContent>
        <extension base="string">
            <attribute name="resourceKey"
                type="string"/>
        </extension>
    </simpleContent>
</complexType>
</element>
<element name="helpURL" minOccurs="0">
<complexType>
    <simpleContent>
        <extension base="string">
            <attribute name="resourceKey"
                type="string"/>
        </extension>
    </simpleContent>
</complexType>
</element>
<element name="group" minOccurs="0">
<complexType>
    <simpleContent>
        <extension base="string">
            <attribute name="resourceKey" type="string"/>
        </extension>
    </simpleContent>
</complexType>
</element>
<element name="wizardClass" type="string" minOccurs="0"/>
</sequence>
<attribute name="name" type="string" use="required"/>
    <attribute name="deprecated" type="boolean" use="optional"/>
</complexType>

<complexType name="Params">
<sequence>
    <element name="param" minOccurs="1" maxOccurs="unbounded">
        <complexType>
            <attribute name="name" type="string" use="required"/>
            <attribute name="type" type="tns:XpathType"
                use="required"/>
            <attribute name="minOccurs" type="string"
                default="1"/>
            <attribute name="maxOccurs" type="string"
                default="1"/>
        </complexType>
    </element>
</sequence>

```

```

        <attribute name="wizardEnabled" type="boolean"
            default="false"/>
    </complexType>
</element>
</sequence>
</complexType>
<simpleType name="XPathType">
    <restriction base="string">
        <enumeration value="string"/>
        <enumeration value="boolean"/>
        <enumeration value="number"/>
        <enumeration value="node-set"/>
        <enumeration value="tree"/>
    </restriction>
</simpleType>
</schema>

```

B.7.1 How to Implement User-Defined XPath Extension Functions

This section describes how to implement user-defined XPath extension functions for Oracle SOA Suite components.

B.7.1.1 How to Implement Functions for the XSLT Mapper

Implementation of user-defined XPath extension functions for the XSLT Mapper is different than for other components:

- Each XSLT Mapper function requires a corresponding public static method from a public static class. The function name and method name must match.
- XSLT Mapper function namespaces must take the form `http://www.oracle.com/XSL/Transform/java/mypackage.MyFunctionClass`, where `mypackage.MyFunctionClass` is the fully qualified class name of the public static class containing the public static methods for the functions.

For additional details about creating a user-defined XPath extension function for the XSLT Mapper, see [Section 45.3.4.4, "Importing User-Defined Functions"](#).

B.7.1.2 How to Implement Functions for All Other Components

For Oracle BPEL Process Manager, Oracle Mediator, and human workflow functions, you must implement either the `oracle.fabric.common.xml.xpath.IXPathFunction` interface (defined in the `fabric-runtime.jar` file) or `javax.xml.xpath.XPathFunction`.

To implement functions for all other components:

1. Implement the `oracle.fabric.common.xml.xpath.IXPathFunction` interface for your XPath function. The `IXPathFunction` interface has one method named `call(context, args)`. The signature of this method is as follows:

```

package oracle.fabric.common.xml.xpath;
public interface IXPathFunction
{
    /** Call this function.
     *
     * @param context The context at the point in the
     *     expression when the function is called.
     * @param args List of arguments provided during
     *     the call of the function.
     */
}

```

```

    */
    public Object call(IXPathContext context, List args) throws
XPathFunctionException;
}

```

where:

- context - The context at the point in the expression when the function is called
- args - The list of arguments provided during the call of the function

For the following example, a function named `getNodeValue(arg1)` is implemented that gets a value of `w3c` node:

```

package com.collaxa.cube.xml.xpath.dom.functions;
import oracle.fabric.common.xml.xpath.IXPathFunction;
import oracle.fabric.common.xml.xpath.IXPathFunction
. . .

public class GetNodeValue implements IXPathFunction {
    Object call(IXPathContext context, List args) throws XPathFunctionException
    {
        org.w3c.dom.Node node = (org.w3c.dom.Node) args.get(0);
        return node.getNodeValue()
    }
}

```

B.7.2 How to Configure User-Defined XPath Extension Functions

This section describes how to configure user-defined XPath extension functions.

To configure user-defined xpath extension functions:

1. Create an XPath extension configuration file in which to define the function. [Example B-2](#) shows a sample configuration file that follows the function schema shown in [Example B-1](#) on page B-60. In this example, two functions are created: `mf:myFunction1` and `mf:myFunction2`.

Example B-2 Sample XML Extension Configuration File

```

<?xml version="1.0" encoding="UTF-8"?>
<soa-xpath-functions resourceBundle="myPackage.myResourceBundle"
xmlns="http://xmlns.oracle.com/soa/config/xpath"
xmlns:mf="http://www.my-functions.com">
<function name="mf:myFunction1">
<className>myPackage.myFunctionClass1</className>
<return type="node-set"/>
<params>
<param name="p1" type="node-set" wizardEnabled="true"/>
<param name="p2" type="string"/>
<param name="p3" type="number" minOccurs="0"/>
<param name="p4" type="boolean" minOccurs="0" maxOccurs="3"/>
</params>
<desc resourceKey="func1-desc-key">this is my first function</desc>
<detail resourceKey="func2-long-desc-key">my first function does ... </detail>
<icon>myPackage/resource/image/myFunction1.png</icon>
<group resourceKey="func-group-key">My Function Group</group>
<wizardClass>myPackage.myWizardClass1</wizardClass>
</function>
<function name="mf:myFunction2">
<className>myPackage.myFunctionClass2</className>

```

```

<return type="string"/>
<params>
  <param name="p1" type="node-set" wizardEnabled="true"/>
  <param name="p2" type="string"/>
  <param name="p3" type="number" minOccurs="0"/>
  <param name="p4" type="boolean" minOccurs="0" maxOccurs="unbounded"/>
</params>
<desc resourceKey="func2-desc-key">this is my second function</desc>
<detail resourceKey="func2-long-desc-key">my second function does ...</detail>
<icon>myPackage/resource/image/myFunction2.png</icon>
<group resourceKey="func-group-key">My Function Group</group>
<wizardClass>myPackage.myWizardClass2</wizardClass>
</function>
</soa-xpath-functions>

```

Table B–1 describes the elements of the configuration file. Each function configuration file uses `soa-xpath-functions` as its root element. The root element has an optional `resourceBundle` attribute. The `resourceBundle` value is the fully qualified class name of the resource bundle class providing NLS support for all function configurations.

Table B–1 Function Schema Elements

Element	Description
<code>className</code>	The fully qualified class name of the function implementation class.
<code>return</code>	The return type of the function. This can be one of the following types supported by XPath and XSLT: string, number, boolean, node-set, and tree.
<code>params</code>	<p>The parameters of the function. A function can have no parameters. A parameter has the following attributes:</p> <ul style="list-style-type: none"> ■ <code>name</code>: The name of the parameter. ■ <code>type</code>: The type of the parameter. This can be one of the following types supported by XPath and XSLT: string, number, boolean, node-set, and tree. ■ <code>minOccurs</code>: The minimum occurrences of the parameter. If set to 0, the parameter is optional. If set to 1, the parameter is required. The current restriction is that this attribute must only take a value of either 0 or 1 and that optional parameters must be defined after the required parameters. The default value is 1 if this attribute is absent. ■ <code>maxOccurs</code>: The maximum occurrences of the parameter. If set to unbounded, the parameter can repeat anytime. This can support functions such as XPath 1.0 function <code>concat()</code>, which can take unlimited parameters. The current restriction is that no parameters except the last parameter of the function can have <code>maxOccurs</code> greater than 1 or unbounded. The default value is 1 if this attribute is absent. ■ <code>wizardEnabled</code>: Indicates whether to enable a wizard to enter the parameter value. This supports a user interface where the parameter value must be entered. If set to <code>true</code>, a wizard launch button is rendered next to the parameter value field. The wizard launch button, when pressed, launches a popup wizard to help the user enter the parameter value. The wizard class must be specified later. The default value is <code>false</code> if this attribute is absent, meaning there is no wizard support for the parameter by default.
<code>desc</code>	An optional description of the function. If the <code>resourceKey</code> is present, the description is retrieved from the resource bundle specified earlier on the root element.
<code>detail</code>	An optional longer (detailed) description of the function. If the <code>resourceKey</code> is present, the description is retrieved from the resource bundle specified earlier on the root element.
<code>icon</code>	An optional icon URL of the function. If the <code>resourceKey</code> is present, the icon URL is retrieved from the resource bundle specified earlier on the root element. This is to support a user interface in which the function must be displayed.

Table B-1 (Cont.) Function Schema Elements

Element	Description
helpURL	An optional help HTML URL of the function. If the <code>resourceKey</code> is present, the help URL is retrieved from the resource bundle specified earlier on the root element. This is to support a user interface in which the function help link must be displayed.
group	An optional group name of the function. If the <code>resourceKey</code> is present, the group name is retrieved from the resource bundle specified earlier on the root element. This is to support a user interface where functions must be grouped. If no group name is specified, the function falls into a built-in advanced functions group when being grouped in a user interface.
wizardClass	The fully qualified class name of the wizard class for all parameters that are wizard-enabled. This is to support a user interface in which parameter values must be entered. This wizard class is invoked by wizard launch buttons to help you enter parameter values. If there is no wizard-enabled parameter, this element must be absent. Note: This element is not supported for user-defined functions. Only system functions currently support this feature.

2. Name your user-defined XPath extension configuration file based on the component type with which to use the function. [Table B-2](#) describes the naming conventions to use for user-defined configuration files.

Table B-2 User-Defined Configuration Files

To Use with This Component...	Use This Configuration File Name...
Oracle BPEL Process Manager	<code>ext-bpel-xpath-functions-config.xml</code>
Oracle Mediator	<code>ext-mediator-xpath-functions-config.xml</code>
XSLT Mapper	<code>ext-mapper-xpath-functions-config.xml</code>
Human workflow	<code>ext-wf-xpath-functions-config.xml</code>
All components	<code>ext-soa-xpath-functions-config.xml</code>

3. Place the configuration file inside a JAR file along with the compiled classes. Within the JAR file, the configuration file must be located in the `META-INF` directory. The JAR file does not need to reside in a specific directory.

Note: The `customXPathFunction` jar must be added explicitly as it is not part of the SOA composite.

4. In Oracle JDeveloper, go to **Tools > Preferences > SOA**.
5. Click the **Add** button and select your JAR file.
6. Restart Oracle JDeveloper for the changes to take effect.

The JAR file is automatically added to the JVM's class path to make it available for use.

B.7.3 How to Deploy User-Defined Functions to Runtime

You can deploy user-defined functions to the runtime environment.

To deploy user-defined functions to runtime:

1. Copy the user-defined function JAR files to `BEA_Home/user_projects/domains/domain_name/lib` or a subdirectory of `lib`.

where *domain_name* is the name of the Oracle WebLogic Server domain (for example, *soainfra*).

2. Restart the Oracle WebLogic Server.

Note: As an alternative, you can add the *BEA_Home/user_projects/domains/domain_name/lib* directory into the class loader. This prevents you from having to restart the Oracle WebLogic Server.

Deployment Descriptor Properties

This appendix describes how to define deployment descriptor properties for BPEL process service components.

This appendix includes the following sections:

- [Section C.1, "Introduction to Deployment Descriptor Properties"](#)
- [Section C.2, "Deprecated 10.1.3 Properties"](#)

Note: You cannot specify deployment descriptor properties at runtime.

C.1 Introduction to Deployment Descriptor Properties

Deployment descriptors are BPEL process service component properties used at runtime by Oracle WebLogic Server, Oracle Enterprise Manager, or both. There are two types of properties:

- Configuration
- Partner link binding

C.1.1 How to Define Deployment Descriptor Properties

You define configuration properties and values in the BPEL process service component section of the `composite.xml` file. [Example C-1](#) shows how to define the `inMemoryOptimization` configuration property.

Example C-1 Configuration Property Definition in `composite.xml`

```
...
  <component name="myBPELServiceComponent">
    ....
    <property name="bpel.config.inMemoryOptimization">true</property>
  </component>
```

[Table C-1](#) lists the configuration deployment descriptor properties.

Table C-1 Properties for the configurations Deployment Descriptors

Property Name	Description
<code>completionPersistPolicy</code>	This property configures how the instance data is saved. It can only be set at the BPEL service component level. The following values are available: <ul style="list-style-type: none"> ■ <code>on</code> (default): The completed instance is saved normally. ■ <code>deferred</code>: The completed instance is saved, but with a different thread and in another transaction. ■ <code>faulted</code>: Only the faulted instances are saved. ■ <code>off</code>: No instances of this process are saved.
<code>oneWayDeliveryPolicy</code>	This property sets the persistence policy of the process in the delivery layer. The possible values are: <ul style="list-style-type: none"> ■ <code>async.persist</code>: Messages into the system are saved in the delivery store before being picked up by the engine. ■ <code>async.cache</code>: Messages into the system are saved in memory before being picked up by the engine. ■ <code>sync</code>: The instance-initiating message is not temporarily saved in the delivery layer. The engine uses the save thread to initiate the message.
<code>inMemoryOptimization</code>	Default value is <code>false</code> . This property can only be set to <code>true</code> if it does not have dehydration points. Activities like <code>wait</code> , <code>receive</code> , <code>onMessage</code> , and <code>onAlarm</code> create dehydration points in the process. If this property is set to <code>true</code> , in-memory optimization is attempted on the instances of this process on <code>to-spec</code> queries.
<code>keepGlobalVariables</code>	Specify whether the server can keep global variable values in the instance store when the instance completes: <ul style="list-style-type: none"> ■ <code>false</code> (default): Global variable values are deleted when the instance completes. ■ <code>true</code>: Global variable values are not deleted.
<code>sensorActionLocation</code>	The location of the sensor action XML file. The sensor action XML file configures the action rule for the events.
<code>sensorLocation</code>	The location of the sensor XML file. The sensor XML file defines the list of sensors into which events are logged.

You define partner link binding properties and values in the BPEL process service component section of the `composite.xml` file. [Example C-2](#) shows how to define the `nonBlockingInvoke` partner link binding property.

Example C-2 Property Definition in `composite.xml`

```

...
<component name="myBPELServiceComponent">
  ...
  <property
name="bpel.partnerLink.nonBlockingInvoke.property">propogate</property>
</component>

```

[Table C-2](#) lists the `partnerLinkBinding` deployment descriptor properties.

Table C-2 Properties for the partnerLinkBinding Deployment Descriptors

Property Name	Description
nonBlockingInvoke	Default value is false. When this is set to true, a separate thread is spawned to do the invocation so that the invoke activity does not block the instance.
validateXML	<p>Enables message boundary validation. When set to true, the XML message is validated against the XML schema during a receive activity and an invoke activity for this partner link. If the XML message is invalid, then a <code>bpelx:invalidVariables</code> runtime fault is thrown. This overrides the domain level <code>validateXML</code> property. The following example enables validation for only the <code>StarLoanService</code> partner:</p> <pre><partnerLinkBinding name="StarLoanService"> <property name="wsdlLocation"> http://<hostname>:9700/orabpel/default/StarLoan/Sta rLoan?wsdl</property> <property name="validateXML">true</property> </partnerLinkBinding></pre>

C.1.2 How to Get the Value of a Preference within a BPEL Process

The value of a property can be read by a BPEL process using the XPath extension function `ora:getPreference(myPref)`. This gets the value of `bpel.preference.myPref`.

This function can be used as part of a simple assign statement, used in condition expressions, or used as part of a more complex XPath expression.

C.2 Deprecated 10.1.3 Properties

Table C-3 lists deprecated properties that can no longer be used.

Table C-3 Deprecated Properties

Property	Deployment Descriptor Type	Deprecated for Release...
completionPersistLevel	configurations	11g Release 1
defaultInput	configurations	11g Release 1
initializeVariables	configurations	11g Release 1
loadSchema	configurations	11g Release 1
noAlterWSDL	configurations	11g Release 1
optimizeVariableCopy	configurations	11g Release 1
relaxTypeChecking	configurations	11g Release 1
relaxXPathQName	configurations	11g Release 1
transaction	configurations	10.1.3.4
SLACompletionTime	configurations	11g Release 1
xpathValidation	configurations	11g Release 1
user	configurations	11g Release 1
pw	configurations	11g Release 1
role	configurations	11g Release 1
correlation	partnerLinkBinding	11g Release 1
contentType	partnerLinkBinding	10.1.3

Table C-3 (Cont.) Deprecated Properties

Property	Deployment Descriptor Type	Deprecated for Release...
retryInterval	partnerLinkBinding	Deprecated by the fault policy feature in 10.1.3.3
retryMaxCount	partnerLinkBinding	Deprecated by the fault policy feature in 10.1.3.3
wSDLLocation	partnerLinkBinding	11g Release 1
wSDLRuntimeLocation	partnerLinkBinding	11g Release 1
wsseHeaders	partnerLinkBinding	11g Release 1
wsseUsername	partnerLinkBinding	11g Release 1
wssePassword	partnerLinkBinding	11g Release 1
preferredPort	partnerLinkBinding	11g Release 1
fullWSAddressing	partnerLinkBinding	11g Release 1

Understanding Sensor Public Views and the Sensor Actions XSD

This appendix describes the available sensor public views and the sensor actions XSD file that you can import into Oracle BPEL Designer.

This appendix includes the following sections:

- [Section D.2, "Sensor Public Views"](#)
- [Section D.3, "Sensor Actions XSD File"](#)

For more information, see [Chapter 17, "Using Oracle BPEL Process Manager Sensors."](#)

D.1 Introduction to Sensor Public Views and the Sensor Actions XSD File

A set of public views is exposed to allow SQL access to sensor values from literally any application interested in the data. In addition, a sample sensor action schema is provided for importing into Oracle BPEL Designer.

D.2 Sensor Public Views

The sensor framework of Oracle BPEL Process Manager provides the functionality to persist sensor values created by processing BPEL instances in a relational schema stored in the dehydration store of Oracle BPEL Process Manager. The data is used to display the sensor values of a process instance in Oracle Enterprise Manager Fusion Middleware Control Console.

D.2.1 BPM Schema

The database publisher persists the sensor data in a predefined relational schema in the database. The following public views can be used from a client (Oracle Warehouse, portals, and so on) to query the sensor values using SQL.

Note: In [Table D-1](#) through [Table D-4](#), the Indexed or Unique? column provides unique index names and the order of the attributes. For example, *U1,2* means that the attribute is the second one in a unique index named *U1*. *PK* means primary key.

D.2.1.1 BPEL_PROCESS_INSTANCES

This view provides an overview of all the process instances of Oracle BPEL Process Manager.

Table D-1 BPEL_PROCESS_INSTANCES View

Attribute Name	SQL Type	Attribute Size	Indexed or Unique?	Null?	Comment
INSTANCE_KEY	NUMBER	--	PK	N	Unique instance ID
APPLICATION_NAME	VARCHAR2	500	--	N	User-defined application name
COMPOSITE_NAME	VARCHAR2	500	--	N	User-defined composite name
REVISION	VARCHAR2	50	--	N	User-defined revision number
LABEL	VARCHAR2	500	--	N	User-defined label
COMPONENT_NAME	VARCHAR2	500	--	N	User-defined component name
TITLE	NVARCHAR2	200	--	Y	User-defined title of the BPEL process
STATE	NUMBER	--	--	Y	State of the BPEL process instance
STATE_TEXT	VARCHAR2	21	--	Y	Text presentation of the state attribute
PRIORITY	NUMBER	--	--	Y	User-defined priority of the BPEL process instance
STATUS	NVARCHAR2	200	--	Y	User-defined status of the BPEL process
STAGE	VARCHAR2	100	--	Y	User-defined stage property of a BPEL process
CONVERSATION_ID	VARCHAR2	256	--	Y	User-defined conversation ID of a BPEL process
CREATION_DATE	TIMESTAMP	6	--	N	Creation time stamp of the process instance
MODIFY_DATE	TIMESTAMP	6	--	Y	Time stamp when the process instance was modified
TS_DATE	DATE	--	--	Y	Date portion of modify_date
TS_HOUR	NUMBER	--	--	Y	Hour portion of modify_date
EVAL_TIME	NUMBER	--	--	Y	Evaluation time of the process instance in milliseconds

D.2.1.2 BPEL_ACTIVITY_SENSOR_VALUES

This view contains all the activity sensor values of the monitored BPEL processes.

Table D-2 BPEL_ACTIVITY_SENSOR_VALUES View

Attribute Name	SQL Type	Attribute Size	Indexed or Unique?	Null?	Comment
SENSOR_NAME	NVARCHAR2	200	U1,2	N	The name of the sensor that fired
SENSOR_TARGET	NVARCHAR2	512	--	N	The target of the fired sensor
ACTION_NAME	NVARCHAR2	200	U1,3	N	The name of the sensor action
ACTION_FILTER	NVARCHAR2	512	--	Y	The filter of the action
CREATION_DATE	TIMESTAMP	6	--	N	The creation date of the activity sensor value
MODIFY_DATE	TIMESTAMP	6	--	Y	The time stamp of last modification

Table D–2 (Cont.) BPEL_ACTIVITY_SENSOR_VALUES View

Attribute Name	SQL Type	Attribute Size	Indexed or Unique?	Null?	Comment
TS_DATE	DATE	--	--	Y	Date portion of modify_date
TS_HOUR	NUMBER	--	--	Y	Hour portion of modify_date
CRITERIA_SATISFIED	VARCHAR2	1	--	Y	NULL, Y, or N
ACTIVITY_NAME	NVARCHAR2	200	--	N	The name of the BPEL activity
ACTIVITY_TYPE	VARCHAR2	30	--	N	The type of the BPEL activity
ACTIVITY_STATE	VARCHAR2	30	--	Y	The state of the BPEL activity
EVAL_POINT	VARCHAR2	30	--	N	The evaluation point of the activity sensor
ERROR_MESSAGE	NCLOB	--	--	Y	An error message
RETRY_COUNT	NUMBER	--	--	Y	The number of retries of the activity
EVAL_TIME	NUMBER	--	--	Y	Evaluation time of the activity in milliseconds
ID	NUMBER	--	PK	N	Unique ID
INSTANCE_KEY	NUMBER	--	U1,1	N	BPEL process ID
APPLICATION_NAME	VARCHAR2	500	--	N	User-defined application name
COMPOSITE_NAME	VARCHAR2	500	--	N	User-defined composite name
REVISION	VARCHAR2	50	--	N	User-defined revision number
LABEL	VARCHAR2	500	--	N	User-defined label
COMPONENT_NAME	VARCHAR2	500	--	N	User-defined component name

D.2.1.3 BPEL_FAULT_SENSOR_VALUES

This view contains all the fault sensor values.

Table D–3 BPEL_FAULT_SENSOR_VALUES View

Attribute Name	SQL Type	Attribute Size	Indexed or Unique?	Null?	Comment
ID	NUMBER	--	PK	N	Unique ID
INSTANCE_KEY	NUMBER	--	U1,1	N	BPEL process ID
APPLICATION_NAME	VARCHAR2	500	--	N	User-defined application name
COMPOSITE_NAME	VARCHAR2	500	--	N	User-defined composite name
REVISION	VARCHAR2	50	--	N	User-defined revision number
LABEL	VARCHAR2	500	--	N	User-defined label
COMPONENT_NAME	VARCHAR2	500	--	N	User-defined component name
SENSOR_NAME	NVARCHAR2	200	U1,2	N	The name of the sensor that fired

Table D-3 (Cont.) BPEL_FAULT_SENSOR_VALUES View

Attribute Name	SQL Type	Attribute Size	Indexed or Unique?	Null?	Comment
SENSOR_TARGET	NVARCHAR2	512	--	N	The target of the fired sensor
ACTION_NAME	NVARCHAR2	200	U1,3	N	The name of the sensor action
ACTION_FILTER	NVARCHAR2	512	--	Y	The filter of the action
CREATION_DATE	TIMESTAMP	6	--	N	The creation date of the activity sensor value
MODIFY_DATE	TIMESTAMP	6	--	Y	The time stamp of last modification
TS_DATE	DATE	--	--	Y	Date portion of modify_date
TS_HOUR	NUMBER	--	--	Y	Hour portion of modify_date
CRITERIA_SATISFIED	VARCHAR2	1	--	Y	NULL if no action filter specified; Y if action filter is specified and evaluates to true; N otherwise
ACTIVITY_NAME	NVARCHAR2	200	--	N	The name of the BPEL activity
ACTIVITY_TYPE	VARCHAR2	30	--	N	The type of the BPEL activity
MESSAGE	CLOB	--	--	Y	The fault message

D.2.1.4 BPEL_VARIABLE_SENSOR_VALUES

This view contains all the variable sensor values.

Table D-4 BPEL_VARIABLE_SENSOR_VALUES View

Attribute Name	SQL Type	Attribute Size	Indexed or Unique?	Null?	Comment
ID	NUMBER	--	PK	N	Unique ID
INSTANCE_KEY	NUMBER	--	U1,1	N	BPEL process ID
APPLICATION_NAME	VARCHAR2	500	--	N	User-defined application name
COMPOSITE_NAME	VARCHAR2	500	--	N	User-defined composite name
REVISION	VARCHAR2	50	--	N	User-defined revision number
LABEL	VARCHAR2	500	--	N	User-defined label
COMPONENT_NAME	VARCHAR2	500	--	N	User-defined component name
SENSOR_NAME	NVARCHAR2	200	U1,2	N	Name of the sensor that fired
SENSOR_TARGET	NVARCHAR2	512	--	N	Target of the sensor
ACTION_NAME	NVARCHAR2	200	U1,3	N	Name of the action
ACTION_FILTER	NVARCHAR2	512	--	Y	Filter of the action
ACTIVITY_SENSOR	NUMBER	--	--	Y	ID of corresponding activity sensor value
CREATION_DATE	TIMESTAMP	6	--	N	Creation date
TS_DATE	DATE	--	--	N	Date portion of creation_date
TS_HOUR	NUMBER	--	--	N	Hour portion of creation_date
VARIABLE_NAME	NVARCHAR2	512	--	N	The name of the BPEL variable

Table D-4 (Cont.) BPEL_VARIABLE_SENSOR_VALUES View

Attribute Name	SQL Type	Attribute Size	Indexed or Unique?	Null?	Comment
EVAL_POINT	VARCHAR2	30	--	Y	Evaluation point of the corresponding activity sensor
CRITERIA_SATISFIED	VARCHAR2	1	--	Y	NULL, Y, or N
TARGET	NVARCHAR2	512	--	--	--
UPDATER_NAME	NVARCHAR2	200	--	N	The name of the activity or event that updated the variable
UPDATER_TYPE	NVARCHAR2	200	--	N	The type of the BPEL activity or event
SCHEMA_NAMESPACE	NVARCHAR2	512	--	Y	Namespace of variable sensor value
SCHEMA_DATATYPE	NVARCHAR2	512	--	Y	Data type of the variable sensor value
VALUE_TYPE	NUMBER	--	--	N	The value type of the variable (corresponds to java.sql.Types values)
VARCHAR2_VALUE	NVARCHAR2	4000	--	Y	The value of string-like variables
NUMBER_VALUE	NUMBER	--	--	Y	
DATE_VALUE	TIMESTAMP	6	--	Y	User-defined date
DATE_VALUE_TZ	VARCHAR2	10	--	Y	User-defined time zone
BLOB_VALUE	BLOB	--	--	Y	
CLOB_VALUE	CLOB	--	--	Y	

D.3 Sensor Actions XSD File

[Example D-1](#) provides a sample sensor action schema that you can import into Oracle BPEL Designer. This schema is also relevant to custom data publishers.

Example D-1 Sample Sensor Action Schema

```
<?xml version="1.0" encoding="utf-8"?>
<!--
  This schema contains the sensor definition. Sensors monitor data
  and execute callbacks appropriately.
-->
<xsd:schema blockDefault="#all" elementFormDefault="qualified"
  targetNamespace="http://xmlns.oracle.com/bpel/sensor"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:tns="http://xmlns.oracle.com/bpel/sensor">

  <xsd:simpleType name="tSensorActionPublishType">
    <xsd:annotation>
      <xsd:documentation>
        This enumeration lists the possible publishing types for probes.
      </xsd:documentation>
    </xsd:annotation>
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="BpelReportsSchema"/>
    </xsd:restriction>
  </xsd:simpleType>

```

```

        <xsd:enumeration value="JMSQueue"/>
        <xsd:enumeration value="JMSTopic"/>
        <xsd:enumeration value="Custom"/>
    </xsd:restriction>
</xsd:simpleType>

<xsd:complexType name="tSensorActionProperty">
    <xsd:simpleContent>
        <xsd:extension base="xsd:string">
            <xsd:attribute name="name" use="required" type="xsd:string"/>
        </xsd:extension>
    </xsd:simpleContent>
</xsd:complexType>

<!--
    Attributes of a sensor action
-->
<xsd:attributeGroup name="tSensorActionAttributes">
    <xsd:attribute name="name" type="xsd:string" use="optional"/>
    <xsd:attribute name="enabled" type="xsd:boolean" use="optional"
default="true"/>
    <xsd:attribute name="filter" type="xsd:string"/>
    <xsd:attribute name="publishName" type="xsd:string" use="required"/>
    <xsd:attribute name="publishType" type="tns:tSensorActionPublishType"
use="required"/>
    <!--
        the name of the JMS Queue/Topic or custom java API, ignored for other
        publishTypes
    -->
    <xsd:attribute name="publishTarget" type="xsd:string" use="optional"/>
</xsd:attributeGroup>

<!--
    The sensor action type. A sensor action consists:
    + unique name
    + effective date
    + expiration date - Optional. If not defined, the probe is active
        indefinitely.
    + filter (to potentially suppress data publishing even if a sensor marks
        it as interesting). - Optional. If not defined, no filter is
        used.
    + publishName A name of a publisher
    + publishType What to do with the sensor data?
    + publishTarget Name of a JMS Queue/Topic or custom publisher.
    + potentially many sensors.
-->
<xsd:complexType name="tSensorAction">
    <xsd:sequence>
        <xsd:element name="sensorName" type="xsd:string" minOccurs="1"
maxOccurs="unbounded"/>
        <xsd:element name="property" minOccurs="0" maxOccurs="unbounded"
type="tns:tSensorActionProperty"/>
    </xsd:sequence>
    <xsd:attributeGroup ref="tns:tSensorActionAttributes"/>
</xsd:complexType>

<!--
    define a listing of sensor actions in a single document. It might be a good
    idea to
        have one sensor action list per business process.
-->

```

```

-->
<xsd:complexType name="tSensorActionList">
  <xsd:sequence>
    <xsd:element name="action" type="tns:tSensorAction" minOccurs="0"
maxOccurs="unbounded" />
  </xsd:sequence>
</xsd:complexType>

<xsd:simpleType name="tSensorKind">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="fault" />
    <xsd:enumeration value="variable" />
    <xsd:enumeration value="activity" />
  </xsd:restriction>
</xsd:simpleType>

<xsd:complexType name="tActivityConfig">
  <xsd:annotation>
    <xsd:documentation>
      The configuration part of an activity sensor comprises of a mandatory
'evalTime' attribute
      and an optional list of variable configurations
    </xsd:documentation>
  </xsd:annotation>
  <xsd:complexContent>
    <xsd:extension base="tns:tSensorConfig">
      <xsd:sequence>
        <xsd:element name="variable" type="tns:tActivityVariableConfig"
maxOccurs="unbounded" minOccurs="0" />
      </xsd:sequence>
      <xsd:attribute name="evalTime" type="xsd:string" use="required" />
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tAdapterConfig">
  <xsd:annotation>
    <xsd:documentation>
      The configuration part of a adapter activity extends the activity
configuration with additional attributes for adapters
    </xsd:documentation>
  </xsd:annotation>
  <xsd:complexContent>
    <xsd:extension base="tns:tActivityConfig">
      <xsd:attribute name="headerVariable" use="required" type="xsd:string" />
      <xsd:attribute name="partnerLink" use="required" type="xsd:string" />
      <xsd:attribute name="portType" use="required" type="xsd:string" />
      <xsd:attribute name="operation" use="required" type="xsd:string" />
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tVariableConfig">
  <xsd:complexContent>
    <xsd:extension base="tns:tSensorConfig">
      <xsd:attribute name="outputDataType" use="required" type="xsd:string" />
      <xsd:attribute name="outputNamespace" use="required" type="xsd:string" />
      <xsd:attribute name="queryName" use="optional" type="xsd:string" />
    </xsd:extension>
  </xsd:complexContent>

```

```
</xsd:complexType>

<xsd:complexType name="tActivityVariableConfig">
  <xsd:complexContent>
    <xsd:extension base="tns:tVariableConfig">
      <xsd:attribute name="target" type="xsd:string" use="required"/>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tFaultConfig">
  <xsd:complexContent>
    <xsd:extension base="tns:tSensorConfig"/>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tNotificationSvcConfig">
  <xsd:complexContent>
    <xsd:extension base="tns:tActivityConfig">
      <xsd:attribute name="inputVariable" use="required" type="xsd:string"/>
      <xsd:attribute name="outputVariable" use="required" type="xsd:string"/>
      <xsd:attribute name="operation" use="required" type="xsd:string"/>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tSensorConfig">
  <xsd:sequence>
    <xsd:element name="action" type="tns:tInlineSensorAction" minOccurs="0"
maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="tInlineSensorAction">
  <xsd:complexContent>
    <xsd:restriction base="tns:tSensorAction"/>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tSensor">
  <xsd:sequence>
    <xsd:element name="activityConfig" type="tns:tActivityConfig"
minOccurs="0"/>
    <xsd:element name="adapterConfig" type="tns:tAdapterConfig" minOccurs="0"/>
    <xsd:element name="faultConfig" type="tns:tFaultConfig" minOccurs="0"/>
    <xsd:element name="notificationConfig" type="tns:tNotificationSvcConfig"
minOccurs="0"/>
    <xsd:element name="variableConfig" type="tns:tVariableConfig"
minOccurs="0"/>
  </xsd:sequence>
  <xsd:attribute name="sensorName" use="required" type="xsd:string"/>
  <xsd:attribute name="kind" use="required" type="tns:tSensorKind"/>
  <xsd:attribute name="classname" use="required" type="xsd:string"/>
  <xsd:attribute name="target" use="required" type="xsd:string"/>
</xsd:complexType>

<xsd:complexType name="tSensorList">
  <xsd:sequence>
    <xsd:element name="sensor" type="tns:tSensor" minOccurs="0"
maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>
```

```

    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="tRouterData">
    <xsd:sequence>
      <xsd:element name="header" type="tns:tHeaderInfo"/>
      <xsd:element name="payload" type="tns:tSensorData"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="tHeaderInfo">
    <xsd:sequence>
      <xsd:element name="processName" type="xsd:string"/>
      <xsd:element name="processRevision" type="xsd:string"/>
      <xsd:element name="domain" type="xsd:string"/>
      <xsd:element name="instanceId" type="xsd:integer"/>
      <xsd:element name="midTierInstance" type="xsd:string"/>
      <xsd:element name="timestamp" type="xsd:dateTime"/>
      <xsd:element name="sensor" type="tns:tSensor"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="tSensorData">
    <xsd:sequence>
      <xsd:element name="activityData" type="tns:tActivityData" minOccurs="0"/>
      <xsd:element name="faultData" type="tns:tFaultData" minOccurs="0"/>
      <xsd:element name="adapterData" minOccurs="0" type="tns:tAdapterData"/>
      <xsd:element name="variableData" type="tns:tVariableData" minOccurs="0"
maxOccurs="unbounded"/>
      <xsd:element name="notificationData" type="tns:tNotificationData"
minOccurs="0"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="tFaultData">
    <xsd:sequence>
      <xsd:element name="activityName" type="xsd:string"/>
      <xsd:element name="activityType" type="xsd:string"/>
      <xsd:element name="data" type="xsd:anyType" minOccurs="0"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="tActivityData">
    <xsd:sequence>
      <xsd:element name="activityType" type="xsd:string"/>
      <xsd:element name="evalPoint" type="xsd:string"/>
      <xsd:element name="errorMessage" nillable="true" minOccurs="0"
type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>

  <!--
  xml type that will be provided to sensors for variable Datas. Note the
  any element represents variable data.
  -->
  <xsd:complexType name="tVariableData">
    <xsd:sequence>
      <xsd:element name="target" type="xsd:string"/>
      <xsd:element name="queryName" type="xsd:string"/>
      <xsd:element name="updaterName" type="xsd:string" minOccurs="1"/>
    </xsd:sequence>
  </xsd:complexType>

```

```

    <xsd:element name="updaterType" type="xsd:string" minOccurs="1"/>
    <xsd:element name="data" type="xsd:anyType"/>
    <xsd:element name="dataType" type="xsd:integer"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="tNotificationData">
  <xsd:complexContent>
    <xsd:extension base="tns:tActivityData">
      <xsd:sequence>
        <xsd:element name="messageID" type="xsd:string" minOccurs="0"
maxOccurs="unbounded"/>
        <xsd:element name="fromAddress" type="xsd:string" minOccurs="0"/>
        <xsd:element name="toAddress" type="xsd:string" minOccurs="0"/>
        <xsd:element name="deliveryChannel" type="xsd:string" minOccurs="0"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>

</xsd:complexType>
<xsd:complexType name="tAdapterData">
  <xsd:complexContent>
    <xsd:extension base="tns:tActivityData">
      <xsd:sequence>
        <xsd:element name="endpoint" type="xsd:string"/>
        <xsd:element name="direction" type="xsd:string"/>
        <xsd:element name="adapterType" type="xsd:string"/>
        <xsd:element name="priority" type="xsd:string" minOccurs="0"/>
        <xsd:element name="messageSize" type="xsd:string" minOccurs="0"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
<!--
The header of the document contains some metadata.
-->
<xsd:complexType name="tSensorActionHeader">
  <xsd:sequence>
    <xsd:element name="processName" type="xsd:string"/>
    <xsd:element name="processVersion" type="xsd:string"/>
    <xsd:element name="processID" type="xsd:long"/>
    <xsd:element name="midTierInstance" type="xsd:string"/>
  </xsd:sequence>
  <xsd:attribute name="actionName" use="required" type="xsd:string"/>
</xsd:complexType>

<!--
Sensor Action data is presented in the form of a header and potentially many
data elements depending on how many sensors associated to the sensor action
marked the data as interesting.
-->
<xsd:complexType name="tSensorActionData">
  <xsd:sequence>
    <xsd:element name="header" type="tns:tHeaderInfo"/>
    <xsd:element name="payload" type="tns:tSensorData" minOccurs="1"
maxOccurs="1"/>
  </xsd:sequence>
</xsd:complexType>
<!--
<xsd:simpleType name="tActivityEvalPoint">

```

```

    <xsd:restriction>
      <xsd:enumeration value="start" />
      <xsd:enumeration value="complete" />
      <xsd:enumeration value="fault" />
      <xsd:enumeration value="compensate" />
      <xsd:enumeration value="retry" />
    </xsd:restriction>
  </xsd:simpleType>

  <xsd:simpleType name="tNotificationAction">
    <xsd:restriction>
      <xsd:enumeration value="creation" />
      <xsd:enumeration value="statusUpdate" />
    </xsd:restriction>
  </xsd:simpleType>
-->

<!--
  The process sensor value header comprises of a timestamp
  where the sensor was triggered and the sensor metadata
-->
<xsd:complexType name="tProcessSensorValueHeader">
  <xsd:sequence>
    <xsd:element name="timestamp" type="xsd:dateTime" />
    <xsd:element ref="tns:sensor" />
  </xsd:sequence>
</xsd:complexType>

<!--
  Extend tActivityData to include more elements
-->
<xsd:complexType name="tProcessActivityData">
  <xsd:complexContent>
    <xsd:extension base="tns:tActivityData">
      <xsd:sequence>
        <xsd:element name="creationDate" type="xsd:dateTime" minOccurs="0"
maxOccurs="1" />
        <xsd:element name="modifyDate" type="xsd:dateTime" minOccurs="0"
maxOccurs="1" />
        <xsd:element name="evalTime" type="xsd:long" minOccurs="0"
maxOccurs="1" />
        <xsd:element name="retryCount" type="xsd:int" minOccurs="0"
maxOccurs="1" />
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<!--
  Extend tVariableData to include more elements
-->
<xsd:complexType name="tProcessVariableData">
  <xsd:complexContent>
    <xsd:extension base="tns:tVariableData">
      <xsd:sequence>
        <xsd:element name="creationDate" type="xsd:dateTime" minOccurs="0"
maxOccurs="1" />
        <xsd:element name="modifyDate" type="xsd:dateTime" minOccurs="0"
maxOccurs="1" />
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

```

```
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<!--
    Extend tFaultData to include more elements
-->
<xsd:complexType name="tProcessFaultData">
    <xsd:complexContent>
        <xsd:extension base="tns:tFaultData">
            <xsd:sequence>
                <xsd:element name="creationDate" type="xsd:dateTime" minOccurs="0"
maxOccurs="1"/>
                <xsd:element name="modifyDate" type="xsd:dateTime" minOccurs="0"
maxOccurs="1"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<!--
    Extend tAdapterData to include more elements
-->
<xsd:complexType name="tProcessAdapterData">
    <xsd:complexContent>
        <xsd:extension base="tns:tAdapterData">
            <xsd:sequence>
                <xsd:element name="creationDate" type="xsd:dateTime" minOccurs="0"
maxOccurs="1"/>
                <xsd:element name="modifyDate" type="xsd:dateTime" minOccurs="0"
maxOccurs="1"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<!--
    Extend tNotificationData to include more elements
-->
<xsd:complexType name="tProcessNotificationData">
    <xsd:complexContent>
        <xsd:extension base="tns:tNotificationData">
            <xsd:sequence>
                <xsd:element name="creationDate" type="xsd:dateTime" minOccurs="0"
maxOccurs="1"/>
                <xsd:element name="modifyDate" type="xsd:dateTime" minOccurs="0"
maxOccurs="1"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<!--
    Copy of tSensorData type with some modified types.
-->
<xsd:complexType name="tProcessSensorData">
    <xsd:sequence>
        <xsd:element name="activityData" type="tns:tProcessActivityData"
minOccurs="0"/>
        <xsd:element name="faultData" type="tns:tProcessFaultData" minOccurs="0"/>
        <xsd:element name="adapterData" minOccurs="0"
maxOccurs="1"/>
    </xsd:sequence>
</xsd:complexType>

```

```

type="tns:tProcessAdapterData"/>
  <xsd:element name="variableData" type="tns:tProcessVariableData"
minOccurs="0" maxOccurs="unbounded"/>
  <xsd:element name="notificationData" type="tns:tProcessNotificationData"
minOccurs="0"/>
</xsd:sequence>
</xsd:complexType>
<!--
  A single process sensor value comprises of the sensor value metadata
  (sensor and timestamp) and the payload (the value) of the sensor
-->
<xsd:complexType name="tProcessSensorValue">
  <xsd:sequence>
    <xsd:element name="header" type="tns:tProcessSensorValueHeader"/>
    <xsd:element name="payload" type="tns:tProcessSensorData"/>
  </xsd:sequence>
</xsd:complexType>

<!--
  Process instance header.
-->
<xsd:complexType name="tProcessInstanceInfo">
  <xsd:sequence>
    <xsd:element name="processName" type="xsd:string"/>
    <xsd:element name="processRevision" type="xsd:string"/>
    <xsd:element name="domain" type="xsd:string"/>
    <xsd:element name="instanceId" type="xsd:integer"/>
  </xsd:sequence>
</xsd:complexType>

<!--
  The list of sensor values comprises of a process header describing the
  BPEL process with name, cube instance id etc. and a list of sensor values
  comprising of sensor metadata information and sensor values.
-->
<xsd:complexType name="tProcessSensorValueList">
  <xsd:sequence>
    <xsd:element name="process" type="tns:tProcessInstanceInfo" minOccurs="1"
maxOccurs="1"/>
    <xsd:element name="sensorValue" type="tns:tProcessSensorValue" minOccurs="0"
maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>

<!-- The sensor list is the root element of the sensor.xml document in the
  bpel process suitcase and is used to define sensors. -->
<xsd:element name="sensors" type="tns:tSensorList"/>

<!-- A sensor is used to monitor a particular aspect of a bpel process -->
<xsd:element name="sensor" type="tns:tSensor"/>

<!-- The actions element is the root element of the sensorAction.xml document
  in the bpel process suitcase and is used to define sensor actions.
  Sensor actions define how to publish data captured by sensors -->
<xsd:element name="actions" type="tns:tSensorActionList"/>

<!-- actionData elements are produced by the sensor framework and sent to the
  appropriate data publishers when sensors 'fire' -->
<xsd:element name="actionData" type="tns:tSensorActionData"/>

```

```
<!-- This element is used when the client API is used to query sensor values
      stored in the default reports schema -->
<xsd:element name="sensorValues" type="tns:tProcessSensorValueList"/>
</xsd:schema>
```

Oracle BAM Web Services Operations

This appendix is a reference for the operations provided by the Oracle BAM `DataObjectOperations` and `DataObjectDefinition` web services. More information about the Oracle BAM web services is available in [Chapter 36, "Using Oracle BAM Web Services."](#)

This appendix contains the following topics:

- [Section E.1, "DataObjectOperations10131"](#)
- [Section E.2, "DataObjectOperationsByName"](#)
- [Section E.3, "DataObjectOperationsByID"](#)
- [Section E.4, "DataObjectDefinition Operations"](#)
- [Section E.5, "ManualRuleFire Operations"](#)

E.1 DataObjectOperations10131

The following operations are supported by the `DataObjectOperations10131` web service:

- [Section E.1.1, "Batch"](#)
- [Section E.1.2, "Delete"](#)
- [Section E.1.3, "Insert"](#)
- [Section E.1.4, "Update"](#)
- [Section E.1.5, "Upsert"](#)

E.1.1 Batch

Batch performs batch operations on a data object.

E.1.1.1 Request Message

The request message contains the following parameters.

`dataObject` (xsd:string)

Full relative path and name of the data object, for example:

```
/Samples/Employees
```

xmlPayload (xsd:any)

Contains the batch payload for any operations to be performed. For example:

```
<payload>
  <_Employees operation="insert">
    <_Salesperson>Tim Bray</_Salesperson>
    <_Sales_Area>EMEA</_Sales_Area>
    <_Sales_Number>12345</_Sales_Number>
  </_Employees>
  <_Employees operation="update" keys="_Sales_Number">
    <_Salesperson>Tim Bray</_Salesperson>
    <_Sales_Area>EMEA</_Sales_Area>
    <_Sales_Number>12345</_Sales_Number>
  </_Employees>
</payload>
```

E.1.2 Delete

Delete removes a row from the data object.

E.1.2.1 Request Message

The request message contains the following parameters.

dataObject (xsd:string)

Full relative path and name of the data object, for example:

```
/Samples/Employees
```

keysCSV (xsd:string)

Comma separated column IDs that must be used as keys, for example:

```
_Sales_Number, _Sales_Area
```

xmlPayload (xsd:any)

Payload for the where clause to delete rows in a data object. For example:

```
<_Employees>
  <_Sales_Number>12345</_Sales_Number>
</_Employees>
```

E.1.3 Insert

Insert adds rows to the specified data object.

E.1.3.1 Request Message

The request message contains the following parameters.

dataObject (xsd:string)

Full relative path and name of the data object, for example:

```
/Samples/Employees
```

xmlPayload (xsd:any)

The payload is specific to each data object.

```
<_Employees>
  <_Salesperson>Time Bray</_Salesperson>
  <_Sales_Area>EMEA</_Sales_Area>
  <_Sales_Number>12345</_Sales_Number>
</_Employees>
```

E.1.4 Update

Update operation updates existing data with new data in a data object.

E.1.4.1 Request Message

The request message contains the following parameters.

dataObject (xsd:string)

Full relative path and name of the data object, for example:

```
/Samples/Employees
```

keysCSV (xsd:string)

Comma separated column IDs that must be used as keys, for example:

```
_Sales_Number, _Sales_Area
```

xmlPayload (xsd:any)

Payload for the update statement and where clause to update rows in a data object. For example:

```
<_Employees>
  <_Sales_Area>Asia Pacific</_Sales_Area>
  <_Sales_Number>12345</_Sales_Number>
</_Employees>
```

E.1.5 Upsert

Upsert operation updates existing data with new data in an existing row in a data object. If the row does not exist a new row is created.

E.1.5.1 Request Message

The request message contains the following parameters.

dataObject (xsd:string)

Full relative path and name of the data object, for example:

```
/Samples/Employees
```

keysCSV (xsd:string)

Comma separated column IDs that must be used as keys, for example:

```
_Sales_Number, _Sales_Area
```

xmlPayload (xsd:any)

Payload for the insert or update statement and where clause to upsert rows in a data object. For example:

```
<_Employees>
  <_Salesperson>Time Bray</_Salesperson>
  <_Sales_Area>EMEA</_Sales_Area>
  <_Sales_Number>12345</_Sales_Number>
</_Employees>
```

E.2 DataObjectOperationsByName

The following operations are supported by the DataObjectOperations10131, DataObjectOperationsByName, and DataObjectOperationsByID web services.

- [Section E.2.1, "Delete"](#)
- [Section E.2.2, "Get"](#)
- [Section E.2.3, "Insert"](#)
- [Section E.2.4, "Update"](#)
- [Section E.2.5, "Upsert"](#)

E.2.1 Delete

Delete removes a row from the data object.

E.2.1.1 Request Message

The request message contains the following parameters.

keysCSV (xsd:string)

Comma separated column IDs that must be used as keys, for example:

```
Sales Number, Sales Area
```

xmlPayload (xsd:any)

Payload for the where clause to delete rows in a data object. For example:

```
<DataObject Name="Employees" Path="/Samples">
  <Contents>
    <Row>
      <Column Name="Salesperson" Value="Greg Guan" />
    </Row>
  </Contents>
</DataObject>
```

E.2.2 Get

Get fetches the details from a data object per the specifications in the XML payload

Get is only available in DataObjectOperationsByName web service.

E.2.2.1 Request Message

The request message contains the following parameters.

keysCSV (xsd:string)

Comma separated column IDs that must be used as keys, for example:

Sales Number, Sales Area

xmlPayload (xsd:string)

The payload specifies what to get from the data object.

For the DataObjectOperationsByName web service the data object name is specified in the payload, for example:

```
<DataObject Name="Employees" Path="/Samples">
  <Contents>
    <Row>
      <Column Name="Salesperson" Value="Greg Masters"/>
    </Row>
  </Contents>
</DataObject>
```

E.2.3 Insert

Insert adds rows to the specified data object.

E.2.3.1 Request Message

The request message contains the following parameters.

xmlPayload (xsd:any)

The payload is specific to each data object.

```
<DataObject Name="Employees" Path="/Samples">
  <Contents>
    <Row>
      <Column Name="Salesperson" Value="Greg Guan" />
      <Column Name="Sales Area" Value="Northeast" />
      <column Name="Sales Number" Value="5671" />
    </Row>
  </Contents>
</DataObject>
```

E.2.4 Update

Update operation updates existing data with new data in a data object.

E.2.4.1 Request Message

The request message contains the following parameters.

keysCSV (xsd:string)

Comma separated column IDs that must be used as keys, for example:

Sales Number, Sales Area

xmlPayload (xsd:any)

Payload for the update statement and where clause to update rows in a data object. For example:

```
<DataObject Name="Employees" Path="/Samples">
  <Contents>
    <Row>
      <Column Name="Salesperson" Value="Greg Guan" />
    </Row>
  </Contents>
</DataObject>
```

E.2.5 Upsert

Upsert operation updates existing data with new data in an existing row in a data object. If the row does not exist a new row is created.

E.2.5.1 Request Message

The request message contains the following parameters.

keysCSV (xsd:string)

Comma separated column IDs that must be used as keys, for example:

Sales Number, Sales Area

xmlPayload (xsd:any)

Payload for the insert or update statement and where clause to upsert rows in a data object. For example:

```
<DataObject Name="Employees" Path="/Samples">
  <Contents>
    <Row>
      <Column Name="Salesperson" Value="Greg Guan" />
      <Column Name="Sales Area" Value="Northeast" />
      <column Name="Sales Number" Value="5671" />
    </Row>
  </Contents>
</DataObject>
```

E.3 DataObjectOperationsByID

The following operations are supported by the DataObjectOperations10131, DataObjectOperationsByName, and DataObjectOperationsByID web services.

- [Section E.3.1, "Batch"](#)
- [Section E.3.2, "Delete"](#)
- [Section E.3.3, "Insert"](#)
- [Section E.3.4, "Update"](#)
- [Section E.3.5, "Upsert"](#)

E.3.1 Batch

Batch performs batch operations on a data object.

E.3.1.1 Request Message

The request message contains the following parameters.

dataObject (xsd:string)

Full relative path and name of the data object, for example:

```
/Samples/Employees
```

xmlPayload (xsd:any)

Contains the batch payload for any operations to be performed. For example:

```
<payload>
  <_Employees operation="insert">
    <_Salesperson>Tim Bray</_Salesperson>
    <_Sales_Area>EMEA</_Sales_Area>
    <_Sales_Number>12345</_Sales_Number>
  </_Employees>
  <_Employees operation="update" keys="_Sales_Number">
    <_Salesperson>Tim Bray</_Salesperson>
    <_Sales_Area>EMEA</_Sales_Area>
    <_Sales_Number>12345</_Sales_Number>
  </_Employees>
</payload>
```

E.3.2 Delete

Delete removes a row from the data object.

E.3.2.1 Request Message

The request message contains the following parameters.

dataObject (xsd:string)

This parameter is not required by the DataObjectOperationsByName web service because the data object name and path are part of the payload.

Full relative path and name of the data object, for example:

```
/Samples/Employees
```

keysCSV (xsd:string)

Comma separated column IDs that must be used as keys, for example:

```
_Sales_Number, _Sales_Area
```

xmlPayload (xsd:any)

Payload for the where clause to delete rows in a data object. For example:

```
<_Employees>
  <_Sales_Number>12345</_Sales_Number>
</_Employees>
```

E.3.3 Insert

Insert adds rows to the specified data object.

E.3.3.1 Request Message

The request message contains the following parameters.

dataObject (xsd:string)

Full relative path and name of the data object, for example:

```
/Samples/Employees
```

xmlPayload (xsd:any)

The payload is specific to each data object.

For the DataObjectOperationsByName web service the data object name is specified in the payload, for example:

```
<_Employees>
  <_Salesperson>Time Bray</_Salesperson>
  <_Sales_Area>EMEA</_Sales_Area>
  <_Sales_Number>12345</_Sales_Number>
</_Employees>
```

E.3.4 Update

Update operation updates existing data with new data in a data object.

E.3.4.1 Request Message

The request message contains the following parameters.

dataObject (xsd:string)

Full relative path and name of the data object, for example:

```
/Samples/Employees
```

keysCSV (xsd:string)

Comma separated column IDs that must be used as keys, for example:

```
_Sales_Number, _Sales_Area
```

xmlPayload (xsd:any)

Payload for the update statement and where clause to update rows in a data object. For example:

```
<_Employees>
  <_Sales_Area>Asia Pacific</_Sales_Area>
  <_Sales_Number>12345</_Sales_Number>
</_Employees>
```

E.3.5 Upsert

Upsert operation updates existing data with new data in an existing row in a data object. If the row does not exist a new row is created.

E.3.5.1 Request Message

The request message contains the following parameters.

dataObject (xsd:string)

Full relative path and name of the data object, for example:

```
/Samples/Employees
```

keysCSV (xsd:string)

Comma separated column IDs that must be used as keys, for example:

```
_Sales_Number, _Sales_Area
```

xmlPayload (xsd:any)

Payload for the insert or update statement and where clause to upsert rows in a data object. For example:

```
<_Employees>
  <_Salesperson>Time Bray</_Salesperson>
  <_Sales_Area>EMEA</_Sales_Area>
  <_Sales_Number>12345</_Sales_Number>
</_Employees>
```

E.4 DataObjectDefinition Operations

The following operations are supported by DataObjectDefinition web service.

- [Section E.4.1, "Create"](#)
- [Section E.4.2, "Delete"](#)
- [Section E.4.3, "Get"](#)
- [Section E.4.4, "Update"](#)

E.4.1 Create

Create creates a new data object. By specifying columnar elements, you can create calculated and lookup fields in addition to regular fields as shown in the examples.

E.4.1.1 Request Message

The request message contains the following parameter.

xmlPayload (xsd:string)

Contains the payload to create a data object.

Table E-1 *xmlPayload Elements and Descriptions and Valid Values*

Element	Description and Values
/DataObject/@External	0 (zero) indicates that the data object is not from an external data source (default). 1 indicates that the data object is from an external data source.
/DataObject/@Name	Name of the data object to be created not including the directory path.
/DataObject/@Path	Directory path in which to create the data object.
/DataObject/@Version	Data objects can be versioned 0 (default) through 14.
/DataObject/@TipText	Description of the data object to be created.
/DataObject/Layout/Column/@Name	Name of the column (field) in the data object.
/DataObject/Layout/Column/@Type	The following values are valid for column type: auto-incr-integer boolean calculated clob datetime decimal float iterID integer lookup string timestamp
/DataObject/Layout/Column/@Nullable	1 (default) indicates that the column supports null values. 0 (zero) indicates that the column does not support null values.
/DataObject/Layout/Column/@Public	1 (default) indicates that the column is public. 0 (zero) indicates that the column is not public.
/DataObject/Layout/Column/@MaxSize	For string type columns, this attribute specifies the maximum permissible string size. Default value is 30.
/DataObject/Layout/Column/@Precision	For decimal type columns, this attribute specifies the precision of the decimal value.
/DataObject/Layout/Column/@Scale	For decimal type columns, this attribute specifies the scale of the decimal value.
/DataObject/Layout/Column/@TipText	Column description

Example E-1 *xmlPayload to Create Data Object With Regular Columns*

```
<DataObject Version="14" Name="Employees3" ID="_Employees3" Path="/Samples"
  External="0">
  <Layout>
    <Column Name="Salesperson" ID="_Salesperson" Type="string" MaxSize="30"
      Nullable="1" Public="1" />
    <Column Name="Sales Number" ID="_Sales_Number" Type="decimal"
      Nullable="1" Public="1" />
    <Column Name="Timestamp" ID="_Timestamp" Type="timestamp"
      Nullable="0" Public="1" />
  </Layout>
</DataObject />
```

```

</Layout>
</DataObject>

```

Example E-2 *xmIPayload to Create Data Object With Lookup Field*

```

<DataObject Version="14" Name="LookupDO" ID="_LookupDO" Path="/Samples">
  <Layout>
    <Description><![CDATA[Lookup]]></Description>
    <Column Name="Name" ID="_Name" Type="string" MaxSize="100"
      Nullable="1" Public="1" />
    <Column Name="ID" ID="_ID" Type="integer" Nullable="1" Public="1" />
    <Column Name="Sales Area" ID="_Sales_Area" Type="lookup">
      <Lookup>
        <DataObject>
          <ID>_Employees</ID>
          <Path>/Samples</Path>
        </DataObject>
        <LookupFieldID>_Sales_Area</LookupFieldID>
        <MatchFields>
          <KeyPair>
            <PrimaryKeyID>_Sales_Number</PrimaryKeyID>
            <ForeignKeyID>_ID</ForeignKeyID>
          </KeyPair>
        </MatchFields>
      </Lookup>
    </Column>
    <Indexes />
  </Layout>
</DataObject>

```

Note that when you construct the XML payload for the Create operation, and the data object version is lower than 12, use `PrimaryKey` instead of `PrimaryKeyID`, `ForeignKey` instead of `ForeignKeyID`, `LookupField` instead of `LookupFieldID`, and provide name values instead of IDs for those fields.

Example E-3 *xmIPayload to Create Data Object With Calculated Field*

```

<DataObject Version="14" Name="CalculatedDO" ID="_CalculatedDO" Path="/Samples">
  <Layout>
    <Description><![CDATA[Calculated Column]]></Description>
    <Column Name="Name" ID="_Name" Type="string" MaxSize="100" Nullable="1"
      Public="1" />
    <Column Name="Address" ID="_Address" Type="string" MaxSize="100" Nullable="1"
      Public="1" />
    <Column Name="Salary" ID="_Salary" Type="decimal" Scale="10" Nullable="1"
      Public="1" />
    <Column Name="Income Tax" ID="_Income_Tax" Type="calculated"
      CalculatedExpression="&lt;expression type='MathExpression'&gt;
&gt;&lt;operation&gt;&lt;left&gt;&lt;type&gt;FieldID&lt;/type&gt;&lt;ivalue&gt;
_Salary&lt;/ivalue&gt;&lt;/left&gt;&lt;operator&gt;*&lt;/operator&gt;&lt;right&gt;
&lt;type&gt;DECIMAL&lt;/type&gt;&lt;ivalue&gt;0.3&lt;/ivalue&gt;&lt;/right&gt;&lt;
/operation&gt;&lt;/expression&gt;" ExpressionUserText="(Salary * 0.3)" />
    <Indexes />
  </Layout>
</DataObject>

```

E.4.1.2 Response Message

void

E.4.2 Delete

Delete removes a data object definition and its contents.

E.4.2.1 Request Message

The request message contains the following parameter.

dataObjectFullName (xsd:string)

Full relative path and name of the data object to be deleted. For example:

```
/Samples/Employees
```

E.4.2.2 Response Message

void

E.4.3 Get

Get retrieves an existing data object definition.

E.4.3.1 Request Message

The request message contains the following parameters.

dataObjectFullName (xsd:string)

Full relative path and name of the data object, for example:

```
/Samples/Sales
```

E.4.3.2 Response Message

The response message contains the following parameter.

xmlPayload (xsd:string)

An XML description of the data object is returned. The schema used is the same definition as described for the Create and Update operations. You can use this operation to find the ID values of the data object and any columns.

Example E-4 xmlPayload for Get Operation

```
<DataObject Version="14" Name="Employees" Path="/Samples" External="0">
  <Layout>
    <Column Name="Salesperson" ID="_Salesperson" Type="string" MaxSize="100"
      Nullable="1" Public="1" />
    <Column Name="Sales Area" ID="_Sales_Area" Type="string" MaxSize="100"
      Nullable="1" Public="1" />
    <Column Name="Sales Number" ID="_Sales_Number" Type="integer" Nullable="1"
      Public="1" />
    <Column Name="Timestamp" ID="_Timestamp" Type="timestamp" Nullable="0" />
    <Indexes />
  </Layout>
</DataObject>
```

E.4.4 Update

Update updates the definition of an existing data object. If a specified column exists in the original definition, the new column definition overwrites the old one. If columns in the existing definition are not specified in the new definition, their definitions are removed. The data object index definition can be updated as well.

E.4.4.1 Request Message

The request message contains the following parameters.

xmlPayload (xsd:string)

Payload for the Update operation is similar to the Create payload with one additional attribute. For example:

```
<DataObject Version="14" Name="Employees4" ID="_Employees4" Path="/Samples"
External="0">
  <Layout>
    <Column Name="Salesperson" ID="_Salesperson" Type="string" MaxSize="50"
      Nullable="1" Public="1" />
    <Column Name="Sales Number" ID="_Sales_Number" Type="integer"
      Nullable="1" Public="1" />
    <Column Name="Timestamp" ID="_Timestamp" Type="timestamp"
      Nullable="0" Public="1" />
    <Indexes />
  </Layout>
</DataObject>
```

E.4.4.2 Response Message

void

E.5 ManualRuleFire Operations

The following operation is supported by ManualRuleFire web service.

- [Section E.5.1, "FireRuleByName"](#)

E.5.1 FireRuleByName

Use this operation to manually launch a rule.

This web service takes a string parameter, which should have user name, followed by a period (.), followed by the alert name. For example:

```
user_name.alertname
```

The period is used as a separator between the user name and the alert name. The web service always treats last period in the string as the separator, allowing the user name to contain periods. For example

```
user.nema.alerrtname
```

It follows then that the alert names cannot contain a period. If you must use the ManualRuleFire web service with an alert containing a period in its name, the alert must be renamed so that it does not contain any periods.

E.5.1.1 Request Message

The request message contains the following parameter.

xmlPayload (xsd:string)

An example:

```
<FireRuleByName xmlns="http://xmlns.oracle.com/bam">
  <strRuleName>string</strRuleName>
</FireRuleByName>
```

E.5.1.2 Response Message

Returns (xsd:string)

```
<FireRuleByNameResponse xmlns="http://xmlns.oracle.com/bam">
  <FireRuleByNameResult>string</FireRuleByNameResult>
</FireRuleByNameResponse>
```

Oracle BAM Alert Rule Options

The following sections describe the options for creating alert rules:

- [Section F.1, "Events"](#)
- [Section F.2, "Conditions"](#)
- [Section F.3, "Actions"](#)
- [Section F.4, "Frequency Constraint"](#)

F.1 Events

Events launch the rule and trigger the action. Each rule contains only one event. Oracle BAM provides the following events:

- [In a specific amount of time](#)
- [At a specific time today](#)
- [On a certain day at a specific time](#)
- [Every interval between two times](#)
- [Every date interval starting on certain date at a specific time](#)
- [When a report changes](#)
- [When a data field changes in data object](#)
- [When a data field in a report meets specified conditions](#)
- [When a data field in a data object meets specified conditions](#)
- [When this rule is launched](#)

F.1.1 In a specific amount of time

When you select the event **In a specific amount of time**, you must complete the rule expression by selecting a time interval in seconds, minutes, or hours.

F.1.2 At a specific time today

When you select the event **At a specific time today**, you must complete the rule expression by selecting the time at which to launch the alert.

F.1.3 On a certain day at a specific time

When you select the event **On a certain day at a specific time**, you must complete the rule expression by selecting both the date and the time at which to launch the alert.

F.1.4 Every interval between two times

When you select the event **Every interval between two times**, you must complete the rule expression by configuring the following settings.

- **select time interval**
Set the number of minutes, hours, or days between each alert launch.
- **select time**
Set the times of day between which the rule is valid and the alert is launched.

F.1.5 Every date interval starting on certain date at a specific time

When you select the event **Every date interval starting on a certain date at a specific time**, you must complete the rule expression by configuring the following settings.

- **select date interval**
Set the alert to launch every Day, Week, Month, or Year.
- **select date**
Set the date on which the rule is valid and the alert is launched.
- **select time**
Set the time of day at which the rule is valid and the alert is launched.

F.1.6 When a report changes

When a report changes is launched when runtime changes in a report occur (not changes in the report definition), that is every time a change list is delivered to the report from the Oracle BAM Server. Report changes can include changes to data in data objects and changes due to Active Now settings.

When you select the event **When a report changes**, you must complete the rule expression by configuring the following settings.

- **select report**
Select the report to monitor for changes.
- **run as <user_name>** (This option appears only if the user creating the alert is a member of the administrator role.)
Select the Oracle BAM user who the selected report runs as. You can select only one run as user. The default run as user is the logged in Oracle BAM user who is creating the alert.
Only recipients who have security permissions that are the same or higher than the run as user receive the notification for report changes, honoring row level security as implemented by the Oracle BAM Architect in the data objects used in the report.
Names that are preceded with a hash (#) are distribution lists.
If there are changes in a report's data object rows that none of the alert recipients have permissions to access, no recipients are notified.

F.1.7 When a data field changes in data object

When you select the event **When a data field changes in a data object**, you must complete the rule expression by configuring the following settings.

Note: The event **When a data field in a data object meets specified conditions** responds only to row inserts and row updates, but it does not respond to row deletes; however, the event **When a data field changes in a data object** responds to row deletes.

- **select data field**

Select the data object field to monitor for changes. In the Field Selection dialog box, locate the data object in the top left section of the dialog box, then select the field in the top right section of the dialog box. Finally, select one or more fields to group by and an aggregate function for the selected field.

- run as **<user_name>** (This option appears only if the user creating the alert is a member of the administrator role.)

Select the Oracle BAM user who the selected report runs as. You can select only one run as user. The default run as user is the logged in Oracle BAM user who is creating the alert.

Only recipients who have security permissions that are the same or higher than the run as user receive the notification for report changes, honoring row level security as implemented by the Oracle BAM Architect in the data objects used in the report.

Names that are preceded with a hash (#) are distribution lists.

If there are changes in a report's data object rows that none of the alert recipients have permissions to access, no recipients are notified.

F.1.8 When a data field in a report meets specified conditions

When you select the event **When a data field changes in a data object**, you must complete the rule expression by configuring the following settings.

- **select report**

Select the report to monitor for changes.

- **this data field has a condition of x**

In the Alert Rule Editor dialog box, select the data object to monitor. Then you can set the condition under which the alert should fire.

- **Row Filter** - Create a filter on a field in the data object to express a condition that, when met, launches the rule. All of the functionality available in report filters is provided. See "Filtering Data" in *Oracle Fusion Middleware User's Guide for Oracle Business Activity Monitoring* for more information.
- **Group Filter** - The Group Filter is similar to the Row Filter in that it provides all of the filtering functionality available in report filters. The special feature here is that it allows filters to be created on a field where a summary function has been applied. See "Filtering Data" in *Oracle Fusion Middleware User's Guide for Oracle Business Activity Monitoring* for more information about building filter expressions.
- **Group** - Choose one or more fields on which to create a grouping, adding further complexity to any filters created in the Row Filter or Group Filter tabs.
- run as **<user_name>** (This option appears only if the user creating the alert is a member of the administrator role.)

Select the Oracle BAM user who the selected report runs as. You can select only one run as user. The default run as user is the logged in Oracle BAM user who is creating the alert.

Only recipients who have security permissions that are the same or higher than the run as user receive the notification for report changes, honoring row level security as implemented by the Oracle BAM Architect in the data objects used in the report.

Names that are preceded with a hash (#) are distribution lists.

If there are changes in a report's data object rows that none of the alert recipients have permissions to access, no recipients are notified.

F.1.9 When a data field in a data object meets specified conditions

When you select the event **When a data field in a data object meets specified condition**, you must complete the rule expression by configuring the following settings.

Note: The event **When a data field in a data object meets specified conditions** responds only to row inserts and row updates, but it does not respond to row deletes; however, the event **When a data field changes in a data object** responds to row deletes.

- **this data field has a condition of x**

In the Alert Rule Editor dialog box, select the data object to monitor. Then you can set the condition under which the alert should fire.

- **Row Filter** - Create a filter on a field in the data object to express a condition that, when met, launches the rule. All of the functionality available in report filters is provided. See "Filtering Data" in *Oracle Fusion Middleware User's Guide for Oracle Business Activity Monitoring* for more information.
- **Group Filter** - The Group Filter is similar to the Row Filter in that it provides all of the filtering functionality available in report filters. The special feature here is that it allows filters to be created on a field where a summary function has been applied. See "Filtering Data" in *Oracle Fusion Middleware User's Guide for Oracle Business Activity Monitoring* for more information about building filter expressions.
- **Group** - Choose one or more fields on which to create a grouping, adding further complexity to any filters created in the Row Filter or Group Filter tabs.
- run as **<user_name>** (This option appears only if the user creating the alert is a member of the administrator role.)

Select the Oracle BAM user who the selected report runs as. You can select only one run as user. The default run as user is the logged in Oracle BAM user who is creating the alert.

Names that are preceded with a hash (#) are distribution lists.

Only recipients who have security permissions that are the same or higher than the run as user receive the notification for report changes, honoring row level security as implemented by the Oracle BAM Architect in the data objects used in the report.

If there are changes in a report's data object rows that none of the alert recipients have permissions to access, no recipients are notified.

F.1.10 When this rule is launched

The event **When this rule is launched** is used to create a rule dependent on another rule which uses the **Launch a rule** action. Several rules can be created using **When this rule is launched** in a hierarchy.

F.2 Conditions

Conditions are optional settings for constraining the time period in which the alert is fired. You can select any number and combination of conditions. Oracle BAM provides the following conditions:

- [If it is between two times](#)
- [If It is between two days](#)
- [If it is a particular day of the week](#)

F.2.1 If it is between two times

Select two times between which the rule should launch.

F.2.2 If It is between two days

Select two dates between which the rule should launch.

F.2.3 If it is a particular day of the week

Select a day of the week on which the rule should launch.

F.3 Actions

Actions are the results of the conditions and events of the rule expression having been met. You can configure any number and combination of actions. Oracle BAM provides the following actions:

- [Send a report via email](#)
- [Send a message via email](#)
- [Send a report via email and escalate to another user after a specific amount of time](#)
- [Send a parameterized message](#)
- [Launch a rule](#)
- [Launch rule if an action fails](#)
- [Delete rows from a Data Object](#)
- [Call a Web Service](#)
- [Run an Oracle Data Integrator Scenario](#)

F.3.1 Send a report via email

Select a report, select to send the report as a report link or as a rendered report, and select a recipient.

F.3.2 Send a message via email

Create an email message to send and select a recipient.

F.3.3 Send a report via email and escalate to another user after a specific amount of time

Select a report to send to the specified user. Select a secondary recipient to receive the message if the first recipient does not respond within the specified time period. The secondary recipient can be a single user or a distribution list.

When the condition of the alert rule is met, a report link is sent to the recipient. To respond to this alert, the recipient must click the report link and view the report. If the recipient does not view the report, it is escalated to the secondary user (or distribution list).

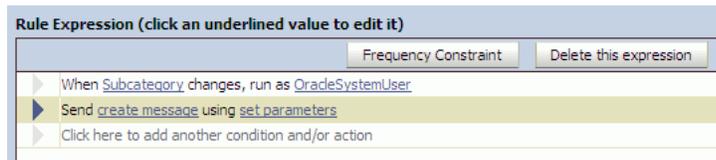
F.3.4 Send a parameterized message

This option enables you to email reports that require parameter inputs to Oracle BAM users. This action enables you to create a fully configurable email message and the parameter values that are passed to the report.

For information about creating prompts and parameters in Oracle BAM dashboards see "Using Prompts and Parameters" in *Oracle Fusion Middleware User's Guide for Oracle Business Activity Monitoring*.

You can use this option to send reports to other users under the conditions specified in the alert message. This action is available for the events **When a data field changes in data object** and **When a data field in a data object meets specified conditions**.

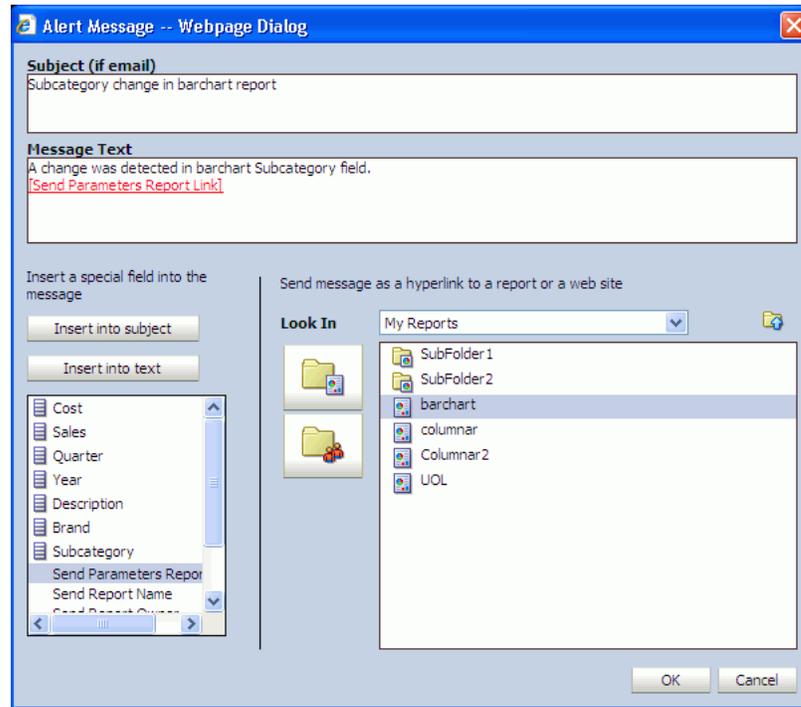
There are two properties that must be configured in this alert action: create message and set parameters.



To create the message

1. Click **create message** in the rule expression.
2. Enter a subject and message to send to the recipient. You can also select links to reports to send in the message body as shown in [Figure F-1](#).

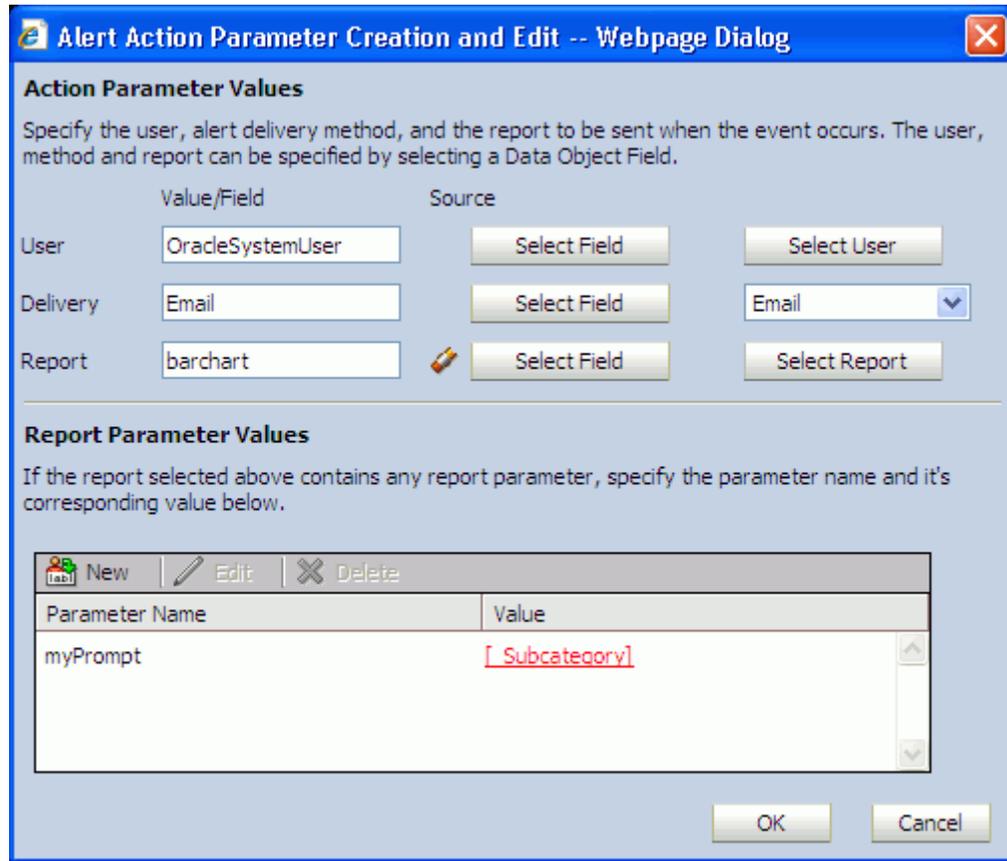
Figure F-1 Alert Message dialog box



To configure the parameter values that are passed to the report when it is opened by the recipient:

1. Click **set parameters** in the rule expression.
2. In the Alert Action Parameter Creation and Edit dialog box, populate the **User**, **Delivery**, and **Report** fields with either predefined values or dynamically from a Data Object field. Use the buttons to set the field values. **Select Field** enables you to select a field in a data object as a value.

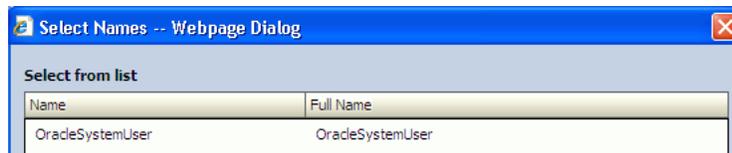
Figure F–2 Alert Action Parameter Creation and Edit dialog box



- **User field**

If you populate this field using the **Select User** button, the recipients are selected from Oracle BAM users listed in Oracle BAM Administrator as shown in [Figure F-3](#).

Figure F–3 Select Names dialog box



- **Delivery field**

If you populate this field with predefined values in the list, the only value that appears in this field is `Email`.

It is not recommended that you use the **Select Field** button as you must then populate a data object with a field set to `smtp` because this is the only delivery method supported. (No other delivery options are supported.)

- **Report field**

If you populate this field with the **Select Report** button, the value that appears in this field is the display name of the report.

If you populate this field from a Data Object, the value must be the report ID of that report, and not the display name. To get the report ID, click the report and click the **Copy Shortcut** link. A window opens with a link such as:

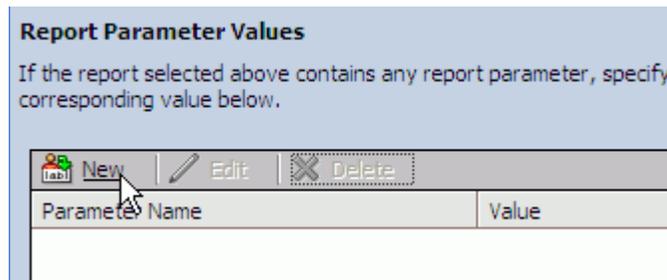
```
http://myServer/oraclebam/ReportServer/default.aspx?Event=ViewReport&ReportDef=1&Buttons=False
```

In this link the **ReportDef** value, 1, is the report ID of the report Emp_Report. Every report in Oracle Business Activity Monitoring has a unique report ID.

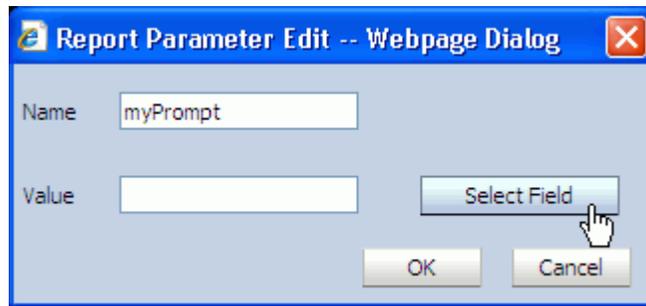
3. Configure the Report Parameter Values.

Enter all of the parameters required by the report.

Click **New** in the **Report Parameter Values** list to configure the parameter.

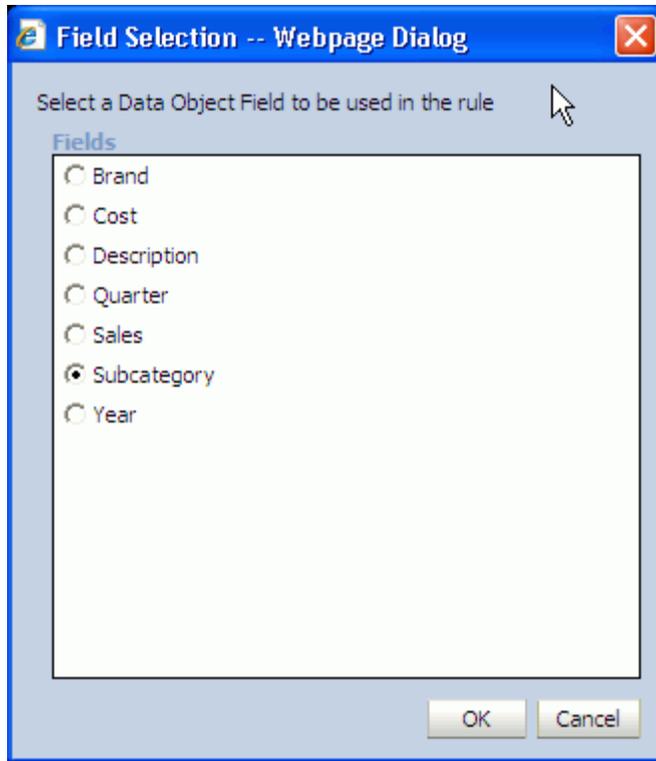


Enter the parameter name in the **Name** field, and click **Select Field** to select the field on which the parameter acts.

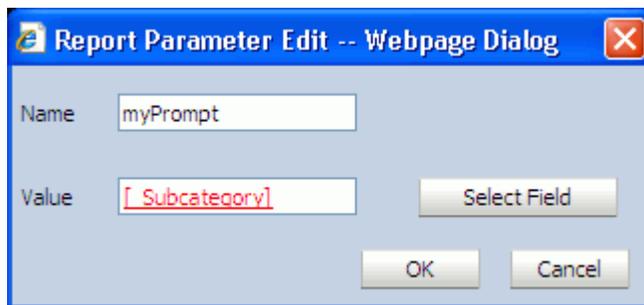


Key in the parameter value, or select the field from the **Field Selection** dialog box, and click **OK**.

For special values use the underscore (_), for example, **_ALL_**, **_BLANK_**, and **_NULL_**.



The selected field ID appears in the **Value** text box. Click **OK** to confirm and return to the parameters list.



F.3.5 Launch a rule

Select a dependent rule that includes the when this rule is launched event. For an example of constructing a dependent rule see [Section 37.5, "Creating Complex Alerts."](#)

F.3.6 Launch rule if an action fails

Select a dependent rule to launch if any of the actions included in the rule fail. For an example of constructing a dependent rule see [Section 37.5, "Creating Complex Alerts"](#)

F.3.7 Delete rows from a Data Object

Select the data object, and construct a filter entry such that when the filter condition is met the row is removed from the data object.

F.3.8 Call a Web Service

When this action is selected, do the following steps to configure the web service:

1. Enter the web service or WSIL end point URL. The URL must begin with the "http" scheme and must end in a valid extension (?WSDL, .WSDL or .WSIL).

For example:

```
http://host_name:port_
number/OracleBAMWS/WebServices/DataObjectOperationsByID?WSDL
```

```
http://api.google.com/GoogleSearch.wsdl
```

```
http://host_name:port_number/inspection.wsil
```

If it is a secure web service select the box and enter the required credentials.

Note: Oracle BAM cannot determine if the web service is hosted on a server which is behind a secure server. It is your responsibility to indicate whether the web service is behind an HTTP basic authentication based server, and you must enter valid credentials if they are required.

2. Click **Display Services** to display the available services of the URL entered in the field.
3. Click **Map Parameters**.

When the event is based on a data object change (for example, [When a data field changes in data object](#), [When a data field in a report meets specified conditions](#), [When a data field in a data object meets specified conditions](#)), a selection list of fields to which the parameter can be mapped is displayed.

To map the parameters choose the **Data Object Field** option, and select a data object field from the list next to each web service parameter listed in the Alert Web Service - Parameter Mapping dialog box.

When the event is not based on a data object change, the value is entered in a text box.

4. Click **OK** to close the Alert Web Service - Parameter Mapping dialog box and the Alert Web Service Configuration dialog box.

Note: If the web service does not respond to the call, then there are no logs available pertaining to the non-response or failure.

F.3.9 Run an Oracle Data Integrator Scenario

Use this action to trigger a scenario in Oracle Data Integrator. This action is only available if the integration files for Oracle Data Integrator have been installed. See [Section 34.2, "Installing the Oracle Data Integrator Integration Files"](#) for more information.

Ensure that the Oracle Data Integrator agent is running and that the agent host, port, and login credentials are properly configured in Oracle Enterprise Manager Fusion Middleware Control. Oracle BAM cannot verify that the Oracle Data Integrator agent is running, and if it is not running, the alert fires, but the action is not carried out as expected. Also, Oracle BAM alerts that trigger Oracle Data Integrator scenarios do not

track the success or failure of the Oracle Data Integrator scenario call, and it is not logged on the Oracle BAM side. See "Configuring Oracle Data Integrator Properties," in *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite* for more information.

In the alert creation dialog box, select the Oracle Data Integrator scenario to invoke by selecting the scenario name and version from the dropdown list.

If the scenario uses variables in it, choose the values (type in a value or choose a field value from the data object) to pass to Scenario Variables in the same screen.

F.4 Frequency Constraint

The Frequency Constraint feature prevents a user's email inbox from being flooded with alerts by limiting the number of alert messages that can be sent out during a given time interval.

Frequency Constraint can be edited only if it is appropriate for the event selected. otherwise it is disabled. It can be set to a value of time which could be in seconds, minutes, or hours.

This limits the number of times the rule launches in a given time period. With real-time data, transactions can occur every millisecond, so alerting frequency must be controlled.

Oracle BAM ICommand Operations and File Formats

This appendix provides a detailed reference for each operation and parameter available in the ICommand command-line utility and web service. This appendix contains the following topics:

- [Section G.1, "Summary of Individual Operations"](#)
- [Section G.2, "Detailed Operation Descriptions"](#)
- [Section G.3, "Format of Command File"](#)
- [Section G.4, "Format of Log File"](#)
- [Section G.5, "Sample Export File"](#)
- [Section G.6, "Regular Expressions"](#)

For more information about ICommand see the following topics:

- [Chapter 38, "Using ICommand"](#)
- [Section 36.5, "Using the ICommand Web Service"](#)

G.1 Summary of Individual Operations

This section summarizes the parameters that can be used with each ICommand operation. You can also see a summary of these operations in the command window by entering `icommand` (without any parameters) at the command prompt.

[Table G-1](#) summarizes the commands available in ICommand.

Table G-1 *ICommand Command Summary*

Command	Parameters
<code>clear</code>	<code>-name <i>itemname</i></code> <code>[-type [dataobject folder distributionlist]]</code> For more information about <code>clear</code> see Section G.2.1, "Clear."

Table G-1 (Cont.) ICommand Command Summary

Command	Parameters
delete	<p>[-name <i>itemname</i>]</p> <p>[-type [dataobject folder report rule securityfilters distributionlist ems eds all]]</p> <p>[-match <i>pattern</i>]</p> <p>[-regex <i>regularexpression</i>]</p> <p>[-all [0 1]]</p> <p>[-systemobjects [0 1]]</p> <p>For more information about delete see Section G.2.2, "Delete."</p>
export	<p>-file <i>file_name</i></p> <p>[-name <i>itemname</i>]</p> <p>[-type [dataobject folder report rule securityfilters distributionlist ems eds all]]</p> <p>[-match <i>pattern</i>]</p> <p>[-regex <i>regularexpression</i>]</p> <p>[-all [0 1]]</p> <p>[-systemobjects [0 1]]</p> <p>[-dependencies [0 1]]</p> <p>[-layout [0 1]]</p> <p>[-contents [0 1]]</p> <p>[-permissions [0 1]]</p> <p>[-owner [0 1]]</p> <p>[-header [0 1]]</p> <p>[-footer [0 1]]</p> <p>[-append [0 1]]</p> <p>[-preview [0 1]]</p> <p>For more information about export see Section G.2.3, "Export."</p>
import	<p>-file <i>file_name</i></p> <p>-continueonerror</p> <p>[-delay <i>milliseconds</i>]</p> <p>[-updatelayout]</p> <p>[-mode [preserveid update overwrite append rename error]]</p> <p>[-preserveowner]</p> <p>[-setcol <i>col_name</i>/[null now value:override_value]]</p> <p>[-preview]</p> <p>For more information about import see Section G.2.4, "Import."</p>
rename	<p>-name <i>itemname</i></p> <p>-newname <i>newitemname</i></p> <p>[-type [dataobject folder report rule distributionlist ems eds]]</p> <p>For more information about rename see Section G.2.5, "Rename."</p>

G.2 Detailed Operation Descriptions

This section details each of the ICommand commands, their parameters, and gives examples. It includes the following topics:

- [Section G.2.1, "Clear"](#)
- [Section G.2.2, "Delete"](#)
- [Section G.2.3, "Export"](#)
- [Section G.2.4, "Import"](#)
- [Section G.2.5, "Rename"](#)

G.2.1 Clear

Clears the contents of an item in the Active Data Cache.

What it means to be *cleared* depends upon the item type:

- For Data Objects, all existing rows within the Data Object are deleted.
- For Folders, all contents of the Folder are deleted.
- For Distribution Lists, all members (users and groups) are removed from the distribution list.

Table G–2 Clear Command Parameters

Parameter	Description
<code>-name <i>itemname</i></code>	The name of the item to be cleared. Required.
<code>-type <i>itemtype</i></code>	The type of the item to be cleared. The following are valid: <ul style="list-style-type: none"> ■ <code>dataobject</code> (see Example G–1) ■ <code>folder</code> ■ <code>distributionlist</code> <code>dataobject</code> is assumed if this parameter is omitted.

Example G–1 Clearing a Data Object

```
icommand -cmd clear -name "/Samples/Call Center" -type dataobject
```

G.2.2 Delete

Deletes an item from the Active Data Cache.

Table G–3 Delete Command Parameters

Parameter	Description
<code>-all [0 1]</code>	Controls whether all items of the specified type are deleted (see Example G–3). A nonzero or omitted value means delete all items of the specified type, a zero (0) value means only delete the named (or matched) items. Zero is assumed if this parameter is omitted.
<code>-match <i>pattern</i></code>	A DOS-style pattern matching string, using the asterisk (*) and question mark (?) characters. The items whose names match the pattern are deleted.
<code>-name <i>itemname</i></code>	The name of the item to be deleted.

Table G-3 (Cont.) Delete Command Parameters

Parameter	Description
<code>-regex <i>regularexpr</i></code>	A regular expression pattern matching string. The items whose names match the pattern are deleted. See Section G.6, "Regular Expressions" for more information.
<code>-systemobjects [0 1]</code>	Controls whether Data Objects in the System folder are included when the <code>all</code> , <code>match</code> , or <code>regex</code> parameters are used. Zero (0) means these data objects are not included. Zero is assumed if this parameter is omitted.
<code>-type <i>itemtype</i></code>	The type of the item to be deleted. The following are valid: <ul style="list-style-type: none"> ▪ <code>dataobject</code> (see Example G-2) ▪ <code>folder</code> ▪ <code>report</code> (see Example G-3) ▪ <code>rule</code> ▪ <code>securityfilters</code> (For the specified Data Objects) ▪ <code>distributionlist</code> ▪ <code>ems</code> (Enterprise Message Source) ▪ <code>eds</code> (External Data Source) ▪ <code>all</code> (see Example G-4) <code>dataobject</code> is assumed if this parameter is omitted.

Example G-2 Deleting a Data Object

```
icommand -cmd delete -name TestDO
//deletes a data object named TestDO. Note that dataobject type is assumed if
the type parameter is not specified.
```

Example G-3 Deleting All Reports

```
icommand -cmd delete -type report -all 1
//deletes all objects of type report
```

Example G-4 Deleting All Objects

```
icommand -cmd delete -type all
//deletes all items except systemobjects
```

G.2.3 Export

Exports information about one or more objects in the Active Data Cache to an XML file. See [Section G.5, "Sample Export File"](#) for an example of an exported data object.

Table G-4 Export Command Parameters

Parameter	Description
-all [0 1]	<p>Controls whether all items of the specified type are exported.</p> <p>A nonzero or omitted value means export all items of the specified type, a zero value means only export the named (or matched) items. Zero (0) is assumed if this parameter is omitted.</p> <p>For Reports, Folders, and Rules, only the items owned by the user running ICommand are exported, unless the user running ICommand is an administrator. When an administrator runs ICommand, any user's items may be exported.</p> <p>See Example G-12, "Exporting All of the Reports in the System"</p>
-append [0 1]	<p>Controls whether the exported information is appended to any existing file.</p> <p>A nonzero value means append. Zero (0) means overwrite the contents of any existing files. Zero is assumed if this parameter is omitted, or if the value is omitted.</p> <p>The Append parameter must be used with the Header and Footer parameters as described in Example G-20, "Using Append Parameter in Export".</p> <p>When the Append parameter is used, the Header and Footer parameters must be defined. If they are not, ICommand includes XML header information and closing XML </OracleBAMExport> tags after each append to the export file. The file is unusable for importing into Oracle BAM, because the import stops when it finds the first </OracleBAMExport> closing tag and ignores the rest of the objects.</p>
-contents [0 1]	<p>Applies only to Data Objects. Controls whether content information (row, column values) is to be exported.</p> <p>A nonzero value means export content information. Zero (0) means do not export content information. nonzero is assumed if this parameter is omitted, or if the value is omitted.</p>
-dependencies [0 1]	<p>Applies to only to Data Objects. Controls whether other Data Objects that the exported Data Objects depend on in the lookup columns are exported.</p> <p>A nonzero value or the parameter present with no value specifies that if the Data Objects being exported contain lookup columns, then the Data Objects that are looked up are exported. Zero is assumed if this parameter is omitted, or if the value is omitted.</p>
-file <i>file_name</i>	<p>The name of the file to export to. Required.</p> <p>If the file does not exist, it is created. If the file does exist, any contents are overwritten, unless the append parameter is used. Because the file contains XML, it usually has an XML extension.</p>

Table G-4 (Cont.) Export Command Parameters

Parameter	Description
-footer [0 1]	<p>Controls whether closing XML information is written to the end of the export file. This can be used to allow successive executions of ICommand to assemble one XML file by repeatedly appending to the same file.</p> <p>A nonzero value means write the closing information. Zero (0) means do not write the closing information. nonzero is assumed if this parameter is omitted, or if the value is omitted.</p> <p>When used with the Append parameter, you must set the Footer value appropriately, or the file cannot be used with ICommand Import. If Footer is not defined, each append includes closing </OracleBAMExport> tags and the import stops when the first closing tag is read and does not import the remaining objects defined in the file.</p> <p>See Example G-20, "Using Append Parameter in Export" for a sample using this parameter.</p>
-header [0 1]	<p>Controls whether XML header information is written to the front of the export file. This can be used to allow successive executions of ICommand to assemble one XML file by repeatedly appending to the same file.</p> <p>A nonzero value means write the header. Zero(0) means do not write the header. nonzero is assumed if this parameter is omitted, or if the value is omitted.</p> <p>See Example G-20, "Using Append Parameter in Export" for a sample using this parameter.</p>
-layout [0 1]	<p>Applies only to Data Objects. Controls whether layout information is to be exported.</p> <p>A nonzero value means export layout information. Zero (0) means do not export layout information. nonzero is assumed if this parameter is omitted, or if the value is omitted.</p>
-match <i>pattern</i>	<p>A DOS-style pattern matching string, using the asterisk (*) and question mark (?) characters. The items whose names match the pattern are exported (see Example G-19, "Exporting a Data Object Using the Match Parameter").</p>
-name <i>itemname</i>	<p>The name of the item to be exported.</p>
-owner [0 1]	<p>Applies only to Folders, Reports, and Rules. Controls whether the information about the owner of the items being exported is included in the export.</p> <p>A nonzero value means export the owner information. Zero (0) means do not export the owner information. nonzero is assumed if this parameter is omitted, or if the value is omitted.</p>
-permissions [0 1]	<p>Applies only to Data Objects and Folders. Controls whether permissions information is to be exported.</p> <p>A nonzero value means export information about the permission settings of the exported Data Objects or Folders. Zero (0) means do not export permission information. Zero is assumed if this parameter is omitted, or if the value is omitted.</p> <p>For Data Objects, only the permissions of the Data Object itself are exported. Any permissions that might be on the folders or subfolders that the Data Objects are contained within are not included.</p> <p>For Folders, the permissions reflect the cumulative permissions of all parent Folders of the Folders being exported.</p>

Table G-4 (Cont.) Export Command Parameters

Parameter	Description
<code>-preview [0 1]</code>	<p>In <code>preview</code> mode, <code>ICommand</code> goes through the motions of exporting all of the specified items, but does not actually output any information. This can be used to see what would be exported for a given command line, and what errors might occur. In this mode, <code>ICommand</code> export continues processing even after some errors that would cause non-<code>preview</code> mode to stop the export.</p> <p>A nonzero value means <code>preview</code> mode. nonzero is assumed if the value is omitted. Zero (0) is assumed if the parameter is omitted.</p>
<code>-regex <i>regexexpr</i></code>	A regular expression pattern matching string. The items whose names match the pattern are exported. See Section G.6, "Regular Expressions" for more information.
<code>-systemobjects [0 1]</code>	Controls whether Data Objects in the System folder are included when the <code>all</code> , <code>match</code> , or <code>regex</code> parameters are used. Zero (0) means these data objects are not included. Zero is assumed if this parameter is omitted.
<code>-type <i>itemtype</i></code>	<p>The type of the item to be exported. The following are valid:</p> <ul style="list-style-type: none"> ▪ <code>dataobject</code> (see Example G-5 and Example G-6) ▪ <code>folder</code> (see Example G-7, Example G-8, and Example G-9) ▪ <code>report</code> (see Example G-10, Example G-11, and Example G-12) ▪ <code>rule</code> (see Example G-13) ▪ <code>securityfilters</code> (For the specified Data Objects) (see Example G-14) ▪ <code>distributionlist</code> (see Example G-15) ▪ <code>ems</code> (Enterprise Message Source) (see Example G-16) ▪ <code>eds</code> (External Data Source) (see Example G-17) ▪ <code>all</code> (see Example G-18) <p><code>dataobject</code> is assumed if this parameter is omitted.</p>

Example G-5 Exporting a Data Object in a Folder

```
icommand -cmd export -name "/Samples/Call Center" -file "C:\CallCenter.xml"
```

Note that the `type` parameter was not included in this example. By default `dataobject` is assigned to `type` if it is not specified.

Example G-6 Exporting a Data Object at the Root

```
icommand -cmd export -name TestDataObject -file "C:\TestDataObject.xml"
```

Note that the data object name was not preceded by the slash (/). When a Data Object is in the root Data Objects folder, a slash is not required.

Example G-7 Exporting a Folder from My Reports

In the first case, the `private:owner/Report` prefix is used in the name parameter because the user exporting the folder is not the folder owner.

```
icommand -cmd export -name "/private:bamadmin/Report/TestMainFolder/TestSubFolder"
-type folder -file C:\FolderExportTest.xml
```

In the second case, the `private:owner/Report` prefix was not used in the name parameter because the user exporting the folder is the folder owner.

```
icommand -cmd export -name "/TestMainFolder/TestSubFolder" -type folder -file  
C:\FolderExportTest.xml
```

Example G–8 Exporting a Folder from Shared Reports

```
icommand -cmd export -name "/public/Report/MainFolderInShared" -type folder -file  
C:\FolderExportTest2.xml
```

Note that the `public` prefix is added to the name parameter.

Example G–9 Exporting a Folder from Data Objects

```
icommand -cmd export -name "/public/DataObject/Test Sub folder" -type folder -file  
C:\foldertest1.xml
```

Example G–10 Exporting a Private Report

As in [Example G–7](#), there are two methods of exporting private reports.

```
icommand -cmd export -name "/private:bamadmin/Report/MyReport" -type report -file  
C:\MyReport.xml
```

```
icommand -cmd export -name MyReport -type report -file C:\MyReport.xml
```

Example G–11 Exporting a Shared Report

```
icommand -cmd export -name "/public/Report/SharedReport" -type report -file  
C:\SharedReport.xml
```

Example G–12 Exporting All of the Reports in the System

```
icommand -cmd export -type report -a11 -file C:\temp\TestAll.xml
```

Example G–13 Exporting an Alert Rule

```
icommand -cmd export -name Alert1 -type rule -file C:\Alert1.xml
```

Example G–14 Exporting a Security Filter

```
icommand -cmd export -type securityfilters -name "TestD0" -file  
"C:\TestFilter.xml"
```

Note that in the name parameter the name of the Data Object is specified rather than the name of the security filter.

Example G–15 Exporting a Distribution List

```
icommand -cmd export -name MyDistList -type distributionlist -file  
C:\MyDistList.xml
```

Example G-16 Exporting an Enterprise Message Source

```
icommand -cmd export -type ems -name TestEMS -file C:\TestEMS.xml
```

Example G-17 Exporting an External Data Source

```
icommand -cmd export -type eds -name TestEDS -file C:\TestEDS.xml
```

Example G-18 Exporting All Oracle BAM Objects in the System

```
icommand -cmd export -type all -file C:\temp\TestAll.xml
```

Example G-19 Exporting a Data Object Using the Match Parameter

```
icommand -cmd export -match "/M*" -file "c:/exportDOstartingwithM.xml"
```

Example G-20 Using Append Parameter in Export

In the first case (the incorrect example), Append is used without setting the Header and Footer parameters (by default Header and Footer are set to 1).

```
icommand -cmd export -type dataobject -name "/Samples/Call Center" -file do.xml
icommand -cmd export -type dataobject -name "/Samples/Employees" -file do.xml
-append
icommand -cmd export -type dataobject -name "/Samples/Film Sales" -file do.xml
-append
```

The output from these commands is as follows. Notice that an XML header and closing tags are included with each append to the file. If this file is used for importing data into Oracle BAM, only the first object is imported. As soon as the first `</OracleBAMExport>` is read at line 4, the import stops.

```
<?xml version="1.0"?>
<OracleBAMExport Version="2020">
  <exported object/>
</OracleBAMExport>
<?xml version="1.0"?>
<OracleBAMExport Version="2020">
  <exported object/>
</OracleBAMExport>
<?xml version="1.0"?>
<OracleBAMExport Version="2020">
  <exported object/>
</OracleBAMExport>
```

In the second case (the correct example), The Header and Footer parameters are specified to produce the necessary output.

```
icommand -cmd export -type dataobject -name "/Samples/Call Center" -file do2.xml
  -header 1 -footer 0
  //only the footer is suppressed in the first command
icommand -cmd export -type dataobject -name "/Samples/Employees" -file do2.xml
  -append 1 -header 0 -footer 0
  //both the header and the footer are suppressed in the intermediate commands
icommand -cmd export -type dataobject -name "/Samples/Film Sales" -file do2.xml
  -append 1 -header 0 -footer 1
  //only the header is suppressed in the last commands
```

The output file produced by these commands can be used to import the objects into an Oracle BAM Server.

```
<?xml version="1.0"?>
<OracleBAMExport Version="2020">
  <exported object>
  <exported object>
</OracleBAMExport>
```

G.2.4 Import

Imports the information from an XML file to an object in the Active Data Cache. The object may be created, replaced, or updated.

If the object does not exist, it is created if possible. For Data Objects, the input file must contain layout information to create the Data Object, and if the file contains no content information, then an empty Data Object is created.

If the user running ICommand is not an administrator, Reports are always imported to the private folders of the user running ICommand. If the path information in the import file exactly matches existing private folders of the user running ICommand, the imported report is placed in that location. Otherwise, it is placed into the root of that user's private folders.

If the user running ICommand is an administrator, then the `preserveowner` option may be used to allow Folders, Reports and Rules to be imported with their original ownership and to their original location.

Table G-5 Import Command Parameters

Parameter	Description
<code>-continueonerror [0 1]</code>	While importing objects from a file, by default, ICommand stops whenever an error is encountered. If you are importing several objects and do not want to stop when an error is found in one, use the <code>continueonerror</code> parameter to continue importing the rest of the objects specified in the command. Specify a one (1) to ignore errors and continue importing other objects (see Example G-21).
<code>-delay <i>millisec</i></code>	Applies only to Data Objects. A value that specifies a delay that is to occur between each row insertion or update. This can be used to simulate active data at a specified rate. The number is the number of milliseconds to wait between each row. It must be greater than zero. If this parameter is omitted, there is no delay. See Example G-21, "Importing a Data Object With Delay"
<code>-file <i>file_name</i></code>	The name of the file to import from. Required. This would usually be a file that was created through the export command.
<code>-preserveowner</code>	Applies only to Folders, Reports, and Rules. Controls whether, when the item is imported, the ownership of the item is set as specified in the import file. This setting of ownership can only be done if the ownership was included in the file during export, and if the user running ICommand is an administrator. A nonzero value means set the ownership as specified in the import file. Zero (0) means the imported items remain owned by the user running ICommand. Zero is assumed if this parameter is omitted, or if the value is omitted.

Table G-5 (Cont.) Import Command Parameters

Parameter	Description
<code>-preview [0 1]</code>	<p>In <code>preview</code> mode, <code>ICommand</code> goes through the motions of importing all of the specified items, but does not actually input any information. This can be used to see what would be imported for a given command line, and what errors might occur. In this mode, <code>ICommand import</code> continues processing even after some errors that would cause non-<code>preview</code> mode to stop the import.</p> <p>A nonzero value means <code>preview</code> mode. nonzero is assumed if the value is omitted. Zero (0) is assumed if the parameter is omitted.</p> <p>This parameter is supported for the following objects: Rule, Distribution list, EDS, EMS, Report, Folder, and Security Filters.</p> <p>See Example G-22, "Importing a Report in Preview mode"</p>

Table G-5 (Cont.) Import Command Parameters

Parameter	Description
-mode <i>mode</i>	<p>By default, if the mode parameter is not specified, the value Error is assumed for objects of type Folder, Report, EDS, EMS, and Distribution List.</p> <p>The following mode values are valid for Folders, Reports, EMS, and EDS objects:</p> <ul style="list-style-type: none"> ■ <code>overwrite</code> If the item exists, replaces it with the imported item. ■ <code>rename</code> If the item exists, changes the name of the imported item. The new name is computed automatically and reported in a message. ■ <code>error</code> If the item exists, terminates the import with an error. <p>The following values are valid for Distribution List objects:</p> <ul style="list-style-type: none"> ■ <code>overwrite</code> If the item exists, replaces it with the imported item. ■ <code>rename</code> If the item exists, changes the name of the imported item. The new name is computed automatically and reported in a message. ■ <code>append</code> If the item exists, appends the users in the imported list to the existing list. ■ <code>error</code> If the item exists, terminates the import with an error. <p>The following value is supported for Data Objects or Reports:</p> <ul style="list-style-type: none"> ■ <code>preserveid</code> This option is important because some other items, such as Reports, point to the Data Objects they use by ID, not by name. <p><u>Data Object Usage:</u> If the imported Data Object does not exist and must be created, ICommand attempts to assign the Data Object the same internal ID that the exported Data Object had. If it cannot, the import is terminated with an error.</p> <p><u>Report Usage:</u> If the imported Report does not exist and must be created, ICommand attempts to assign the Report the same internal ID that the exported Report had. If it cannot, the import is terminated with an error.</p>

Table G-5 (Cont.) Import Command Parameters

Parameter	Description
<code>-mode mode</code> (cont.)	<p>Only the following value is valid for Data Objects:</p> <ul style="list-style-type: none"> ▪ <code>update</code> Typically, when ICommand imports a Data Object, it creates a new Data Object or locates the existing Data Object and inserts the imported rows into that Data Object. In <code>update</code> mode, ICommand instead attempts to locate existing matching rows by Row ID, and updates those existing rows with the values in the import file. Unmatched rows are inserted. For matching Row IDs in the import file that have no data columns specified, the rows are deleted from the existing Data Object. <p>For Security Filters, the only value supported is <code>overwrite</code>. If <code>overwrite</code> is not specified and the Data Object contains at least one Security Filter, the import is terminated with an error.</p> <p>This parameter is not supported for Rules.</p>
<code>-setcol</code>	<p>Allows override of column values from the command line during import, including setting to current date/time.</p> <p><code>-setcol column_name/NULL</code> <code>-setcol column_name/NOW</code> <code>-setcol column_name/VALUE:override-value</code></p> <p><code>column_name</code> is the name of a column in the Data Object being imported. This cannot be a column of type lookup or calculated. Column names that are not contained in the input XML being imported can be specified, if they are columns in the Data Object being imported into.</p> <p>The portion after the slash specifies a value that should be substituted for that column on each row that is imported -- any value for that column in the import file is ignored (overridden). Note that slash is the one character that is not permitted in column names, so there is no potential conflict with any column names in this syntax.</p> <p>NULL specifies that the column value should be set to null. The column must be defined as "nullable" in the Data Object's layout.</p> <p>NOW specifies that the column value should be set to the current date/time when the column value is being set into the row. This option can only be used for columns of type datetime, timestamp, and string.</p> <p>VALUE:override-value specifies an arbitrary constant value (after the colon) that the column should be set to. The value must be a legal value for the type of the column.</p> <p>To allow multiple columns to be overridden, any number of <code>setcol</code> parameters may be present. However, because duplicate parameters are not permitted, ICommand recognizes any parameter name that starts with <code>setcol</code> as a <code>setcol</code> parameter (for example, <code>setcol1</code>, <code>setcol2</code>, and so on).</p> <p>Sample command line:</p> <pre>icommand -cmd import -file myfile.xml -setcol1 Field1/null -setcol2 Field3/now -setcol3 "Customer Name/value:John Q. Public"</pre>

Table G-5 (Cont.) Import Command Parameters

Parameter	Description
-updatelayout	Applies only to Data Objects. Controls whether, if the Data Object being imported exists, the layout (schema) of the Data Object is updated according to the layout information in the import file. True if parameter is present; false if parameter is not present.

Example G-21 Importing a Data Object With Delay

```
icommand -cmd import -file C:\TestDO.xml -delay 1000 -continueonerror 1
```

Example G-22 Importing a Report in Preview mode

```
icommand -cmd import -file C:\TestReport.xml -preview 1
```

G.2.5 Rename

Renames an item in the Active Data Cache.

Table G-6 Rename Command Parameters

Parameter	Description
-name <i>itemname</i>	The name of the item to be renamed. Required. The full folder path must be given when renaming objects of type Folder (see Example G-24, "Renaming Folders").
-newname <i>newitemname</i>	The new name for the item. Required. The full folder path must be given when renaming objects of type Folder (see Example G-24, "Renaming Folders"). For Data Objects and Reports, only the new base name should be given, with no path (for example -newname "MyReport").
-type <i>itemtype</i>	The type of object to be renamed. The following are valid: <ul style="list-style-type: none"> ■ dataobject (see Example G-23) ■ folder (see Example G-24) ■ report (see Example G-25) ■ rule ■ distributionlist (see Example G-26) ■ ems (Enterprise Message Source) ■ eds (External Data Source) dataobject is assumed if this parameter is omitted. all is not supported as an item type in the rename command.

Example G-23 Renaming a Data Object in a Folder

```
icommand -cmd rename -type dataobject -name "/TestDataObjectFolder/TestDataObject"
-newname NewTestDataObject
```

Example G-24 Renaming Folders

Renaming a data object folder:

```
icommand -cmd rename -type folder -name "/public/DataObject/TestFolder"
-newname "/public/DataObject/NewTestFolder"
```

Renaming a private report folder:

```
iccommand -cmd rename -type folder -name "/private:weblogic/Report/MySubFolder"
-newname "/private:weblogic/Report/NewMySubFolder"
```

Renaming a shared report folder

```
iccommand -cmd rename -type folder -name "/public/Report/TestSubFolder"
-newname "/public/Report/NewTestSubFolder"
```

Example G–25 Renaming a Report in a Private Folder

```
iccommand -cmd rename -type report -name "/TestReportFolder/TestReport" -newname
NewTestReport
```

Example G–26 Renaming a Distribution List

```
iccommand -cmd rename -type distributionlist -name TestList -newname MyDistList
```

G.3 Format of Command File

This section contains the following topics:

- [Section G.3.1, "Inline Content"](#)
- [Section G.3.2, "Command IDs"](#)
- [Section G.3.3, "Continue On Error"](#)

The command file contains the root tag `OracleBAMCommands`.

Within the root tag is a tag for every command to be executed. The tag name is the command name, and the parameters for the command are attributes.

Sample command file:

```
<?xml version="1.0" encoding="utf-8"?>
<OracleBAMCommands continueonerror="1">
  <Export name="Samples/Media Sales" file="MediaSales.xml" contents="0" />
  <Rename name="Samples/Call Center" newname="Call Centre" />
  <Delete type="EMS" name="WebLog" />
  <Delete type="EMS" name="WebLog2" />
</OracleBAMCommands>
```

The output of this sample command file is shown in [Section G.4, "Format of Log File."](#)

G.3.1 Inline Content

When using a command file to import, the `inline` option enables you to include the import content inside the command file, rather than in a separate import file. Here is an example:

```
<?xml version="1.0"?>
<OracleBAMCommands>
<Import inline="1">
<OracleBAMExport Version="2013">
  <DataObject Version="14" Name="Employees_Inline" ID="_Employees_Inline"
    Path="/Samples" External="0">
    <Layout>
      <Column Name="Salesperson" ID="_Salesperson" Type="string" MaxSize="100"
        Nullable="1" Public="1"/>
      <Column Name="Sales Area" ID="_Sales_Area" Type="string" MaxSize="100"
```

```

        Nullable="1" Public="1"/>
<Column Name="Sales Number" ID="_Sales_Number" Type="integer"
    Nullable="1" Public="1"/>
<Column Name="Timestamp" ID="_Timestamp" Type="timestamp" Nullable="0"
    Public="1"/>
<Indexes/>
</Layout>
<Contents>
  <Row ID="1">
    <Column ID="_Salesperson" Value="Greg Masters"/>
    <Column ID="_Sales_Area" Value="Northeast"/>
    <Column ID="_Sales_Number" Value="567"/>
    <Column ID="_Timestamp" Value="2004-09-14T14:07:41.0000560PDT"/>
  </Row>
  <Row ID="2">
    <Column ID="_Salesperson" Value="Lynette Jones"/>
    <Column ID="_Sales_Area" Value="Southwest"/>
    <Column ID="_Sales_Number" Value="228"/>
    <Column ID="_Timestamp" Value="2004-09-14T14:07:41.0000560PDT"/>
  </Row>
  <Row ID="3">
    <Column ID="_Salesperson" Value="Noel Rogers"/>
    <Column ID="_Sales_Area" Value="Northwest"/>
    <Column ID="_Sales_Number" Value="459"/>
    <Column ID="_Timestamp" Value="2004-09-14T14:07:41.0000560PDT"/>
  </Row>
</Contents>
</DataObject>
</OracleBAMExport>
</Import>
</OracleBAMCommands>

```

G.3.2 Command IDs

This feature is only used when output is being sent to a log file. To make the parsing of log results easier, each command can be given an ID. This ID is included in the Result or Error elements of any output related to that command.

Sample Input:

```

<OracleBAMCommands continueonerror="1">
  <Delete id="1" type="dataobject" name="Data Object A"/>
  <Delete id="2" type="dataobject" name="Data Object B"/>
</OracleBAMCommands>

```

Sample Output Log File:

```

<?xml version="1.0"?>
<ICommandLog Login="weblogic">
  <Results Command="Delete" ID="1">Data Object &quot;/Data Object A&quot;
  deleted.</Results>
  <Error Command="Delete" ID="2">
    <![CDATA[BAM-02409: There is no Data Object named "Data Object B".
    [ErrorSource="ICommandEngine",ErrorID="ICommandEngine.DOExist"]]]>
  </Error>
</ICommandLog>

```

G.3.3 Continue On Error

Ordinarily, ICommand executes commands in a command file until a failure occurs, or until they all complete successfully. In other words, if a command file contains 20 commands, and the second command fails for any reason, then no further commands are executed. This behavior can be changed by using the `continueonerror` attribute at either a global level or for each command.

[Example G–27](#) shows how to use the `continueonerror` attribute so that all commands are executed regardless of if any failures occur

Example G–27 Enabling Global ContinueOnError Mode

```
<OracleBAMCommands continueonerror="1">
  <Delete id="1" type="dataobject" name="Data Object A"/>
  <Delete id="2" type="dataobject" name="Data Object B"/>
</OracleBAMCommands>
```

In [Example G–28](#), `continueonerror` only applies to the command that deletes Data Object A. If this command fails, then ICommand outputs the error and continues. But if any other command fails, ICommand stops immediately.

Example G–28 Enabling Command-Level ContinueOnError Mode

```
<OracleBAMCommands>
  <Delete id="1" type="dataobject" name="Data Object A" continueonerror="1"/>
  <Delete id="2" type="dataobject" name="Data Object B"/>
  <Delete id="3" type="dataobject" name="Data Object C"/>
  <Delete id="4" type="dataobject" name="Data Object D"/>
</OracleBAMCommands>
```

G.4 Format of Log File

The log file contains the root tag `ICommandLog`.

Within the root tag is an entry for every error or informational message logged.

Errors are logged with the tag `Error`.

Informational messages are logged with the tag `Results`.

Both `Results` and `Error` tags optionally contain an attribute of the form `Command=cmdname`, if appropriate, that contains the name of the command that generated the error or informational message.

This sample log file is output of command file given in [Section G.3, "Format of Command File"](#):

```
<?xml version="1.0" encoding="utf-8"?>
<ICommandLog Login="user_name">
  <Results Command="Export">Data Object "/Samples/Media Sales" exported
  successfully (0 rows).</Results>
  <Results Command="Export">1 items exported successfully.</Results>
  <Results Command="Rename">Data Object "/Samples/Call Center" renamed to
  "/Samples/Call Centre".</Results>
  <Results Command="Delete">Enterprise Message Source "WebLog" deleted.</Results>
  <Error Command="Delete"><![CDATA[Error while processing command "Delete".
  [ErrorSource="ICommand", ErrorID="ICommand.Error"] There is no Enterprise Message
  Source named "WebLog2". [ErrorSource="ICommand",
  ErrorID="ICommand.EMSEist"]]]></Error>
</ICommandLog>
```

G.5 Sample Export File

The following example shows a sample file resulting from exporting a Data Object.

```
<?xml version="1.0"?>
<OracleBAMExport Version="2018">
  <DataObject Version="14" Name="Employees" ID="_Employees" Path="/Samples"
  External="0">
    <Layout>
      <Column Name="Salesperson" ID="_Salesperson" Type="string" MaxSize="100"
      Nullable="1" Public="1"/>
      <Column Name="Sales Area" ID="_Sales_Area" Type="string" MaxSize="100"
      Nullable="1" Public="1"/>
      <Column Name="Sales Number" ID="_Sales_Number" Type="integer" Nullable="1"
      Public="1"/>
      <Column Name="Timestamp" ID="_Timestamp" Type="timestamp" Nullable="0"
      Public="1"/>
      <Indexes/>
    </Layout>
    <Contents>
      <Row ID="1">
        <Column ID="_Salesperson" Value="Greg Masters"/>
        <Column ID="_Sales_Area" Value="Northeast"/>
        <Column ID="_Sales_Number" Value="567"/>
        <Column ID="_Timestamp" Value="2004-09-14T14:07:41.0000560PDT"/>
      </Row>
      <Row ID="2">
        <Column ID="_Salesperson" Value="Lynette Jones"/>
        <Column ID="_Sales_Area" Value="Southwest"/>
        <Column ID="_Sales_Number" Value="228"/>
        <Column ID="_Timestamp" Value="2004-09-14T14:07:41.0000560PDT"/>
      </Row>
      <Row ID="3">
        <Column ID="_Salesperson" Value="Noel Rogers"/>
        <Column ID="_Sales_Area" Value="Northwest"/>
        <Column ID="_Sales_Number" Value="459"/>
        <Column ID="_Timestamp" Value="2004-09-14T14:07:41.0000560PDT"/>
      </Row>
    </Contents>
  </DataObject>
</OracleBAMExport>
```

G.6 Regular Expressions

The `export` and `delete` commands optionally accept a regular expression with the `regex` parameter.

A regular expression is a pattern of text that consists of ordinary characters (for example, letters a through z) and special characters, known as *metacharacters*. The pattern describes one or more strings to match when searching for items by name.

Note: The behavior of `ICommand -regex` is exactly like the `java.util.regex` package for matching character sequences against patterns specified by regular expressions.

Table G–7 contains the complete list of metacharacters and their behavior in the context of regular expressions.

Table G–7 Metacharacters for Regular Expressions

Character	Description
\	Marks the next character as a special character, a literal, a backreference, or an octal escape. For example, 'n' matches the character "n". '\n' matches a newline character. The sequence '\\\' matches "\" and \"(\" matches "(".
^	Matches the position at the beginning of the input string. If the RegExp object's <code>multiline</code> property is set, ^ also matches the position following '\n' or '\r'.
\$	Matches the position at the end of the input string. If the RegExp object's <code>multiline</code> property is set, \$ also matches the position preceding '\n' or '\r'.
*	Matches the preceding character or subexpression zero or more times. For example, <code>zo*</code> matches "z" and "zoo". * is equivalent to <code>{0,}</code> .
+	Matches the preceding character or subexpression one or more times. For example, <code>zo+</code> matches "zo" and "zoo", but not "z". + is equivalent to <code>{1,}</code> .
?	Matches the preceding character or subexpression zero or one time. For example, <code>do(es)?</code> matches the "do" in "do" or "does". ? is equivalent to <code>{0,1}</code> .
{n}	<i>n</i> is a nonnegative integer. Matches exactly <i>n</i> times. For example, <code>o{2}</code> does not match the 'o' in "Bob," but matches the two o's in "food".
{n,}	<i>n</i> is a nonnegative integer. Matches at least <i>n</i> times. For example, <code>o{2,}</code> does not match the "o" in "Bob" and matches all the o's in "foooood". <code>o{1,}</code> is equivalent to <code>o+</code> . <code>o{0,}</code> is equivalent to <code>o*</code> .
{n,m}	<i>M</i> and <i>n</i> are nonnegative integers, where $n \leq m$. Matches at least <i>n</i> and at most <i>m</i> times. For example, <code>o{1,3}</code> matches the first three o's in "foooood". <code>o{0,1}</code> is equivalent to <code>o?</code> . Note that you cannot put a space between the comma and the numbers.
?	When this character immediately follows any of the other quantifiers (<code>*</code> , <code>+</code> , <code>?</code> , <code>{n}</code> , <code>{n,}</code> , <code>{n,m}</code>), the matching pattern is non-greedy. A non-greedy pattern matches as little of the searched string as possible, whereas the default greedy pattern matches as much of the searched string as possible. For example, in the string "oooo", <code>o+?</code> matches a single "o", while <code>o+</code> matches all 'o's.
.	Matches any single character except "\n". To match any character including the '\n', use a pattern such as <code>[\s\S]</code> .
(pattern)	A subexpression that matches <i>pattern</i> and captures the match. The captured match can be retrieved from the resulting Matches collection using the <code>\$0</code> . . . <code>\$9</code> properties. To match parentheses characters (), use <code>\(</code> or <code>\)</code> .
(?:pattern)	A subexpression that matches <i>pattern</i> but does not capture the match, that is, it is a non-capturing match that is not stored for possible later use. This is useful for combining parts of a pattern with the "or" character (). For example, <code>industr(?:y ies)</code> is a more economical expression than <code>industry industries</code> .

Table G-7 (Cont.) Metacharacters for Regular Expressions

Character	Description
<code>(?=pattern)</code>	A subexpression that performs a positive lookahead search, which matches the string at any point where a string matching <i>pattern</i> begins. This is a non-capturing match, that is, the match is not captured for possible later use. For example 'Windows (?=95 98 NT 2000)' matches "Windows" in "Windows 2000" but not "Windows" in "Windows 3.1". Lookaheads do not consume characters, that is, after a match occurs, the search for the next match begins immediately following the last match, not after the characters that comprised the lookahead.
<code>(?!pattern)</code>	A subexpression that performs a negative lookahead search, which matches the search string at any point where a string not matching <i>pattern</i> begins. This is a non-capturing match, that is, the match is not captured for possible later use. For example 'Windows (?!95 98 NT 2000)' matches "Windows" in "Windows 3.1" but does not match "Windows" in "Windows 2000". Lookaheads do not consume characters, that is, after a match occurs, the search for the next match begins immediately following the last match, not after the characters that comprised the lookahead.
<code>x y</code>	Matches either <i>x</i> or <i>y</i> . For example, 'z food' matches "z" or "food". '(z f)ood' matches "zood" or "food".
<code>[xyz]</code>	A character set. Matches any of the enclosed characters. For example, '[abc]' matches the 'a' in "plain".
<code>[^xyz]</code>	A negative character set. Matches any character not enclosed. For example, '[^abc]' matches the 'p' in "plain".
<code>[a-z]</code>	A range of characters. Matches any character in the specified range. For example, '[a-z]' matches any lowercase alphabetic character in the range 'a' through 'z'.
<code>[^a-z]</code>	A negative range characters. Matches any character not in the specified range. For example, '[^a-z]' matches any character not in the range 'a' through 'z'.
<code>\b</code>	Matches a word boundary, that is, the position between a word and a space. For example, 'er\b' matches the 'er' in "never" but not the 'er' in "verb".
<code>\B</code>	Matches a nonword boundary. 'er\B' matches the 'er' in "verb" but not the 'er' in "never".
<code>\cx</code>	Matches the control character indicated by <i>x</i> . For example, \cM matches a Control-M or carriage return character. The value of <i>x</i> must be in the range of A-Z or a-z. If not, <i>c</i> is assumed to be a literal 'c' character.
<code>\d</code>	Matches a digit character. Equivalent to [0-9].
<code>\D</code>	Matches a nondigit character. Equivalent to [^0-9].
<code>\f</code>	Matches a form-feed character. Equivalent to \x0c and \cL.
<code>\n</code>	Matches a newline character. Equivalent to \x0a and \cJ.
<code>\r</code>	Matches a carriage return character. Equivalent to \x0d and \cM.
<code>\s</code>	Matches any white space character including space, tab, form-feed, and so on. Equivalent to [\f\n\r\t\v].
<code>\S</code>	Matches any non-white space character. Equivalent to [^\f\n\r\t\v].
<code>\t</code>	Matches a tab character. Equivalent to \x09 and \cI.

Table G-7 (Cont.) Metacharacters for Regular Expressions

Character	Description
<code>\v</code>	Matches a vertical tab character. Equivalent to <code>\x0b</code> and <code>\cK</code> .
<code>\w</code>	Matches any word character including underscore. Equivalent to <code>[A-Za-z0-9_]</code> .
<code>\W</code>	Matches any nonword character. Equivalent to <code>[^A-Za-z0-9_]</code> .
<code>\xn</code>	Matches <i>n</i> , where <i>n</i> is a hexadecimal escape value. Hexadecimal escape values must be exactly two digits long. For example, <code>'\x41'</code> matches "A". <code>'\x041'</code> is equivalent to <code>'\x04'</code> & "1". Allows ASCII codes to be used in regular expressions.
<code>\num</code>	Matches <i>num</i> , where <i>num</i> is a positive integer. A reference back to captured matches. For example, <code>'(.)\1'</code> matches two consecutive identical characters.
<code>\n</code>	Identifies either an octal escape value or a backreference. If <code>\n</code> is preceded by at least <i>n</i> captured subexpressions, <i>n</i> is a backreference. Otherwise, <i>n</i> is an octal escape value if <i>n</i> is an octal digit (0-7).
<code>\nm</code>	Identifies either an octal escape value or a backreference. If <code>\nm</code> is preceded by at least <i>nm</i> captured subexpressions, <i>nm</i> is a backreference. If <code>\nm</code> is preceded by at least <i>n</i> captures, <i>n</i> is a backreference followed by literal <i>m</i> . If neither of the preceding conditions exists, <code>\nm</code> matches octal escape value <i>nm</i> when <i>n</i> and <i>m</i> are octal digits (0-7).
<code>\nml</code>	Matches octal escape value <i>nml</i> when <i>n</i> is an octal digit (0-3) and <i>m</i> and <i>l</i> are octal digits (0-7).
<code>\un</code>	Matches <i>n</i> , where <i>n</i> is a Unicode character expressed as four hexadecimal digits. For example, <code>\u00A9</code> matches the copyright symbol (©).

Normalized Message Properties

Header manipulation and propagation is a key business integration messaging requirement. Oracle BPEL Process Manager, Oracle Mediator, Oracle JCA, and B2B rely extensively on header support to solve customers' integration needs. For example, you can preserve a file name from the source directory to the target directory by propagating it through message headers. In Oracle BPEL Process Manager and Oracle Mediator, you can access, manipulate, and set headers with varying degrees of UI support.

Normalized Message is simplified to have only two parts, properties and payload.

Typically, properties are name-value pairs of scalar types. To fit the existing complex headers into properties, properties will be flattened into scalar types.

The user experience is simplified while manipulating headers in design time, because the complex properties are predetermined. In the mediator or BPEL designer, you can manipulate the headers with some reserved key words.

However, this method does not address the properties that are dynamically generated based on your input. Based on your choice, the header definitions are defined. These definitions are not predetermined and hence cannot be accounted for in the list of predetermined property definitions. You cannot design header manipulation of the dynamic properties before they are defined. To address this limitation, you must generate all the necessary services (composite entry points) and references. This restriction applies to services that are expected to generate dynamic properties. Once dynamic properties are generated, they must be stored for each composite. Only then you can manipulate the dynamic properties in Mediator or BPEL designer.

The chapter includes the following sections:

- [Section H.1, "Oracle BPEL Process Manager Properties"](#)
- [Section H.2, "Oracle Web Services Addressing Properties"](#)

H.1 Oracle BPEL Process Manager Properties

[Table H-1](#) lists all the predetermined properties of a normalized message for Oracle BPEL Process Manager.

Table H-1 Properties for Oracle BPEL Process Manager

Property Name	Propagatable (Yes/No)	Direction (Inbound /Outbound)	Data Type	Range of Valid Values	Description
<code>bpel.metadata</code>	Yes	Both	String	Any string, size limit: 1000	This contains extra information that user wants to associate the BPEL instance to. Whatever passed in will be stored in the metadata column of the cube_instance table.
<code>bpel.priority</code>	Yes	Inbound	String that can be read into an integer	[1-10]. 1 being the highest priority	Goes into cube_instance priority column. Used by system to prioritize.
<code>bpel.title</code>	No	Inbound	String	Any string, size limit: 100	Goes into the title column of cube_instance table.
<code>bpel.instanceIndex1</code>	No	Inbound	String	Any string, size limit: 100	This goes into ci_indexes table. Extra index for cube_instance.
<code>bpel.instanceIndex2</code>	No	Inbound	String	Any string, size limit: 100	This goes into ci_indexes table. Extra index for cube_instance.
<code>bpel.instanceIndex3</code>	No	Inbound	String	Any string, size limit: 100	This goes into ci_indexes table. Extra index for the cube_instance.

H.2 Oracle Web Services Addressing Properties

[Table H-2](#) lists all the predetermined properties of a normalized message for Web Services Addressing.

Table H-2 Properties for Oracle Web Services Addressing

Property Name	Propagatable (Yes/No)	Direction (Inbound /Outbound)	Data Type	Range of Valid Values	Description
<code>wsa.messageId</code>	No	Both	String	URI format	This property specifies the identifier for the message and the endpoint to which replies to this message should be sent as an Endpoint Reference.
<code>wsa.relatesTo</code>	No	Both	String	URI format	This optional (repeating) element information item contributes one abstract [relationship] property value, in the form of an (IRI, IRI) pair. The content of this element (of type <code>xs:anyURI</code>) conveys the [message id] of the related message.
<code>wsa.replyToAddress</code>	No	Both	String	URI format	Is a contract between two components communicating asynchronously.
<code>wsa.replyToPortType</code>	No	Both	QName	Any QName	This value is passed to the WS service to configure portType on the service's callback. It is translated to the WSA callback EndpointReference's PortType element.

Table H-2 (Cont.) Properties for Oracle Web Services Addressing

Property Name	Propagatable (Yes/No)	Direction (Inbound /Outbound)	Data Type	Range of Valid Values	Description
wsa.replyToService	No	Both	QName	Any QName	This value is passed to the WS service to configure service on the service's callback. It is translated to the WSA callback EndpointReference's ServiceName element.
wsa.action	No	Both	String	URI format	This REQUIRED element (whose content is of type <code>xs:anyURI</code>) conveys the value of the [action] property.
wsa.to	No	Both	String	URI format	This optional element (whose content is of type <code>xs:anyURI</code>) provides the value for the [destination] property. If this element is NOT present then the value of the [destination] property is http://www.w3.org/2005/08/addressing/anonymous .

Oracle User Messaging Service Applications

This appendix describes how to create your own Oracle User Messaging Service applications using the procedures and code provided.

This chapter includes the following sections:

- [Section I.1, "Send Message to User Specified Channel"](#)
- [Section I.2, "Send Email with Attachments"](#)

Note: For more information, and for code samples, refer to Oracle Technology Network (<http://www.oracle.com/technology>).

I.1 Send Message to User Specified Channel

This chapter describes how to build and run the Send Message to User Specified Channel application provided with Oracle User Messaging Service.

Note: To learn about the architecture and components of Oracle User Messaging Service, see *Oracle Fusion Middleware Getting Started with Oracle SOA Suite*.

This chapter contains the following sections:

- [Section I.1.1, "Overview"](#)
- [Section I.1.2, "Installing and Configuring SOA and User Messaging Service"](#)
- [Section I.1.3, "Building the Sample"](#)
- [Section I.1.4, "Creating a New Application Server Connection"](#)
- [Section I.1.5, "Deploying the Project"](#)
- [Section I.1.6, "Configuring User Messaging Preferences"](#)
- [Section I.1.7, "Testing the Sample"](#)

I.1.1 Overview

The "Send Message to User Specified Channel" application demonstrates a BPEL process that allows a message to be sent to a user through a messaging channel specified in User Messaging Preferences. After you have configured a device and messaging channel addresses for each supported channel and the default device,

Oracle User Messaging Service routes the message to the user based on the preferred channel setting that you configured.

I.1.1.1 Provided Files

The following files are included in the application:

- SendMessage.pdf – this document.
- Project – the directory containing Oracle JDeveloper project files.
- Readme.txt.
- Release notes

I.1.2 Installing and Configuring SOA and User Messaging Service

The installation of SOA and User Messaging Service has already been performed on your hosted instance, and the sample users have already been seeded. Perform the following steps to enable notifications in soa-infra, if not already done:

1. Using Enterprise Manager, go to the **SOA Infrastructure** menu, and select **SOA Administration > Workflow Notification Properties**, and set **Notification Mode** to **ALL**.
2. Configure the User Messaging drivers if required as described in "Configuring Drivers" in the *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite*.
3. Set the email address for user `weblogic` by using the JXplorer LDAP browser. Refer to "[Updating Addresses in Your LDAP User Profile](#)".
4. Restart the server.

I.1.2.1 Updating Addresses in Your LDAP User Profile

Perform the following steps to set the email address for user `weblogic` by using the JXplorer LDAP browser:

I.1.2.1.1 Installing Download and install JXplorer from <http://www.jxplorer.org>.

I.1.2.1.2 Connecting 1. Set the embedded LDAP server admin password as follows:

- Login to the Oracle WebLogic Server Administration Console.
- Click the domain name link > **Security > Embedded LDAP**.
- Enter a new **Credential** and **Confirm Credential** (for example, `weblogic`).
- Click **Save**.

2. Connect from JXplorer by specifying the fields in [Table I-1](#):

Table I-1 JXplorer Connection Fields

Field	Value
Host	WLS AdminServer hostname
Port	WLS AdminServer port
Protocol	LDAP v3
Security Level	User + Password
User DN	cn=Admin
Password	password

I.1.2.1.3 Setting User Messaging Device Addresses in LDAP The following example uses the user `weblogic`. You may create and use additional users.

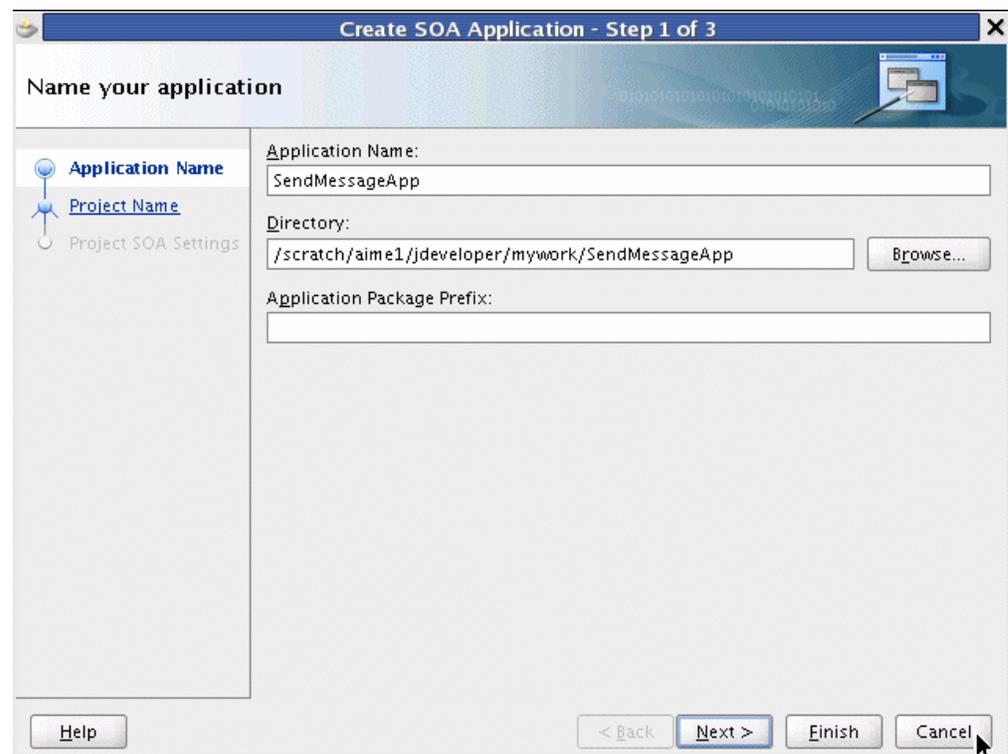
1. Expand the LDAP tree as follows: **domain > myrealm > people > weblogic**.
2. Click the user entry.
3. Select the HTML view tab on the right.
4. Enter the necessary Email Address and Mobile Phone Number.
5. Click **Submit**.

I.1.3 Building the Sample

Performing the following procedure of building the sample from scratch allows you to learn how to add messaging to your SOA Composite Applications, and use User Messaging Preferences.

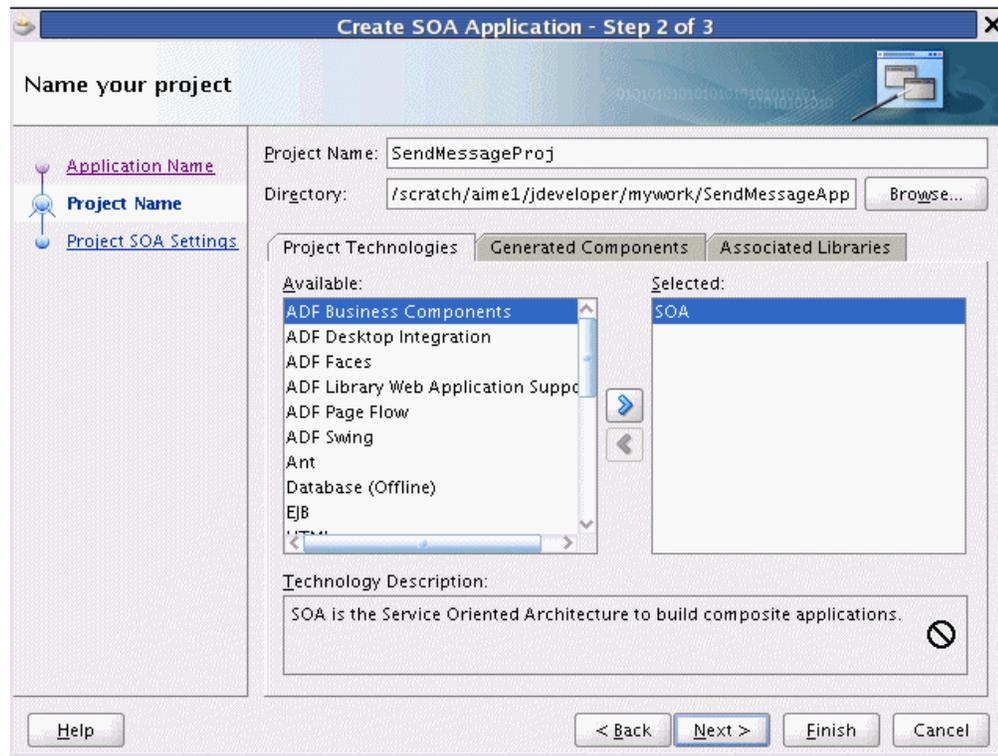
1. Open Oracle JDeveloper 11g.
2. Create a new application by selecting **File, New, General, Applications, and SOA Applications**. Click **OK**.
3. Enter the *Application Name* and click **Next** (Figure I-1).

Figure I-1 Creating a New Application and Project (1 of 3)



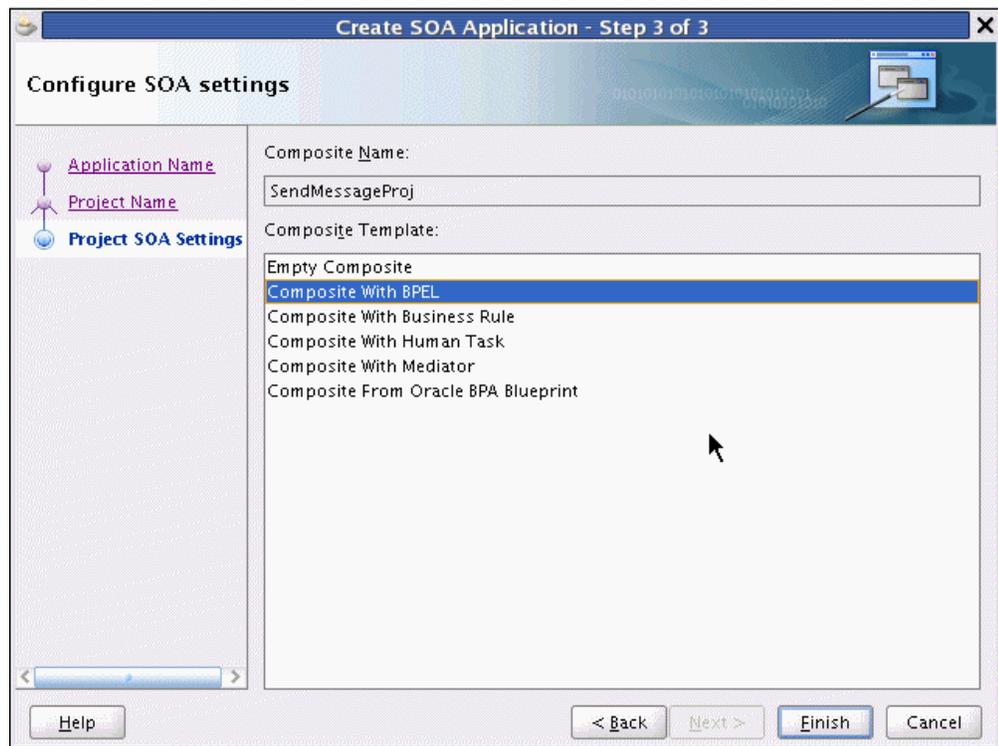
4. Enter the name for the project and click **Next** (Figure I-2).

Figure I-2 Creating a New Application and Project (2 of 3)



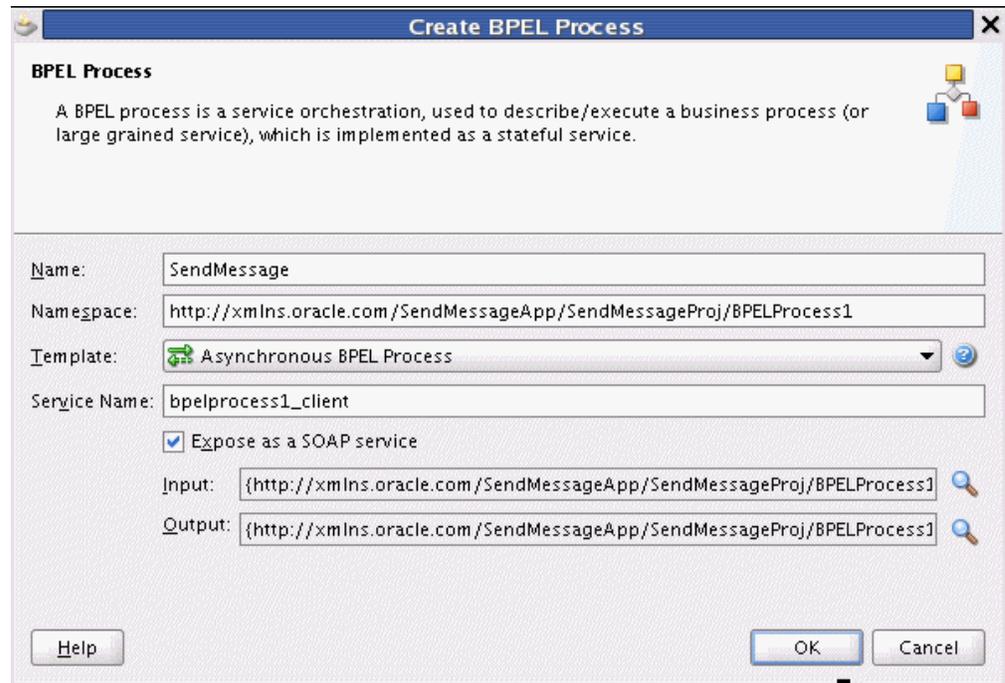
5. Select the **Composite With BPEL** composite template (Figure I-3). Click **Finish**.

Figure I-3 Creating a New Application and Project (3 of 3)



6. In the Create BPEL Process dialog, enter the BPEL process name as `SendMessage` (Figure I-4). Click **OK**.

Figure I-4 Creating the BPEL Process

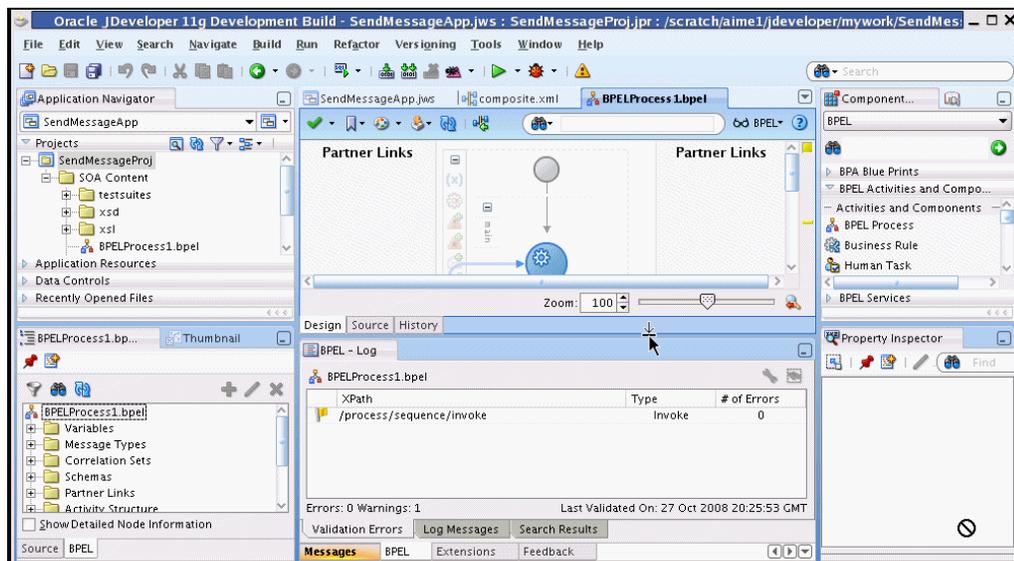


7. Verify that **Expose as a SOAP service** is checked. Click **OK**.
8. You have now created an empty and default BPEL application (Figure I-5).

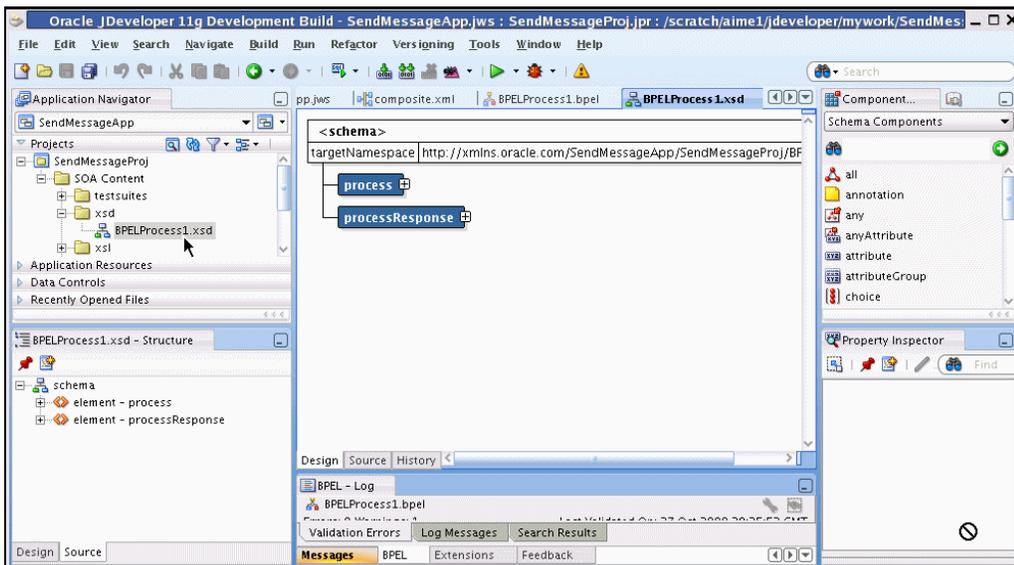
In the Oracle JDeveloper main window you can view the following components of the application under the **Composite.xml** tab.

- The left box is the definition of a web service client that is used to initiate an application.
- The middle box is a BPEL process that creates and formats the message and calls the messaging service.

Note: You will later create the messaging service resource that is used to send the message when you create the User Notification BPEL process (steps 13 - 19).

Figure I-5 Empty and Default BPEL Application

9. Expand the `xsd` folder in the Application Navigator and open `BPELProcess1.xsd` by double-clicking it (Figure I-6).

Figure I-6 Accessing the BPELProcess1.xsd File

10. Click the **Source** tab (Figure I-7).
11. Perform the following modifications to the inputs of this BPEL application:
In the generated file, `SendMessage.xsd`, in the `xsd` folder in the Application Navigator under projects, the following element definition is created by default:

```
<element name="input" type="string"/>
```

This XSD element defines the input for the BPEL process.

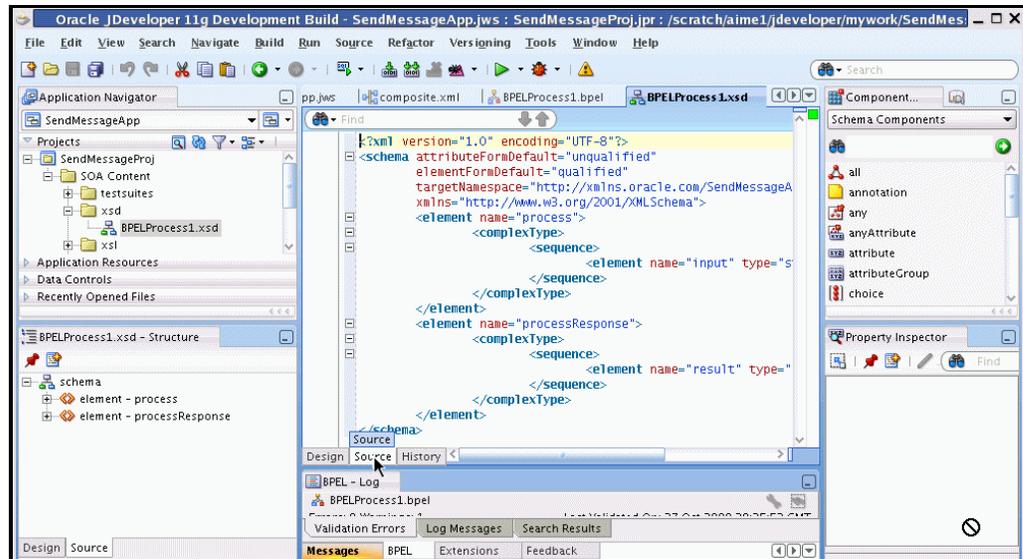
Select the **Source** tab (Figure I-7), and replace the line above with the following three lines:

```

<element name="to" type="string"/>
<element name="subject" type="string"/>
<element name="body" type="string"/>

```

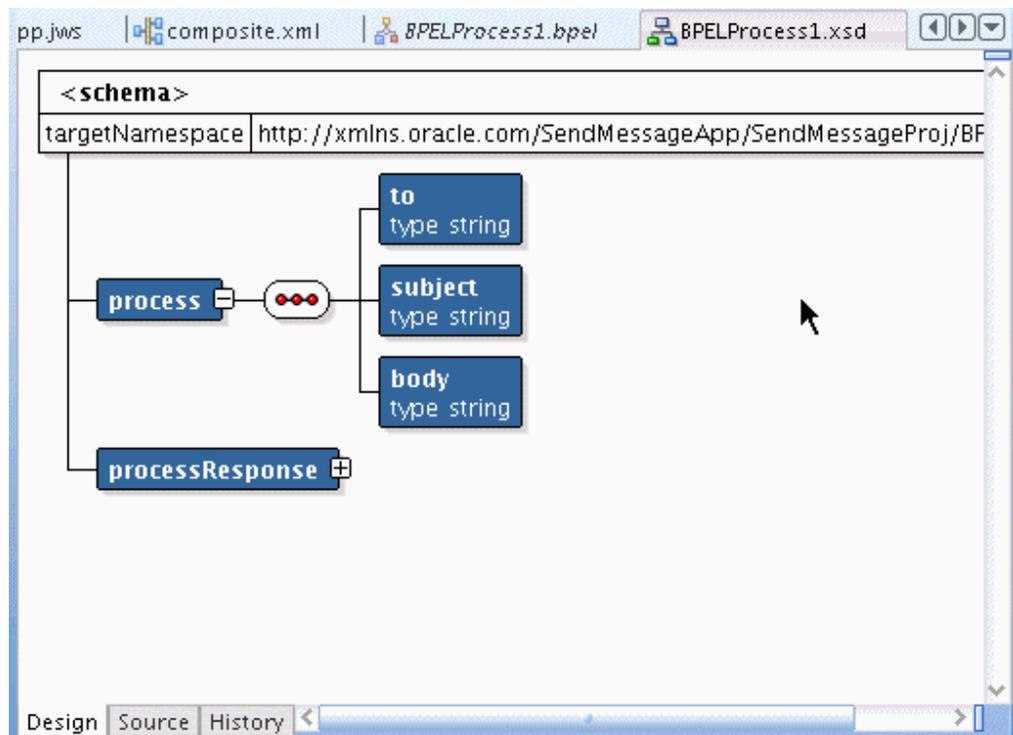
Figure I-7 Modifying the Inputs in the BPELProcess1.xsd File



12. From the **File** menu, select **Save All**.

13. View the expanded process element (Figure I-8).

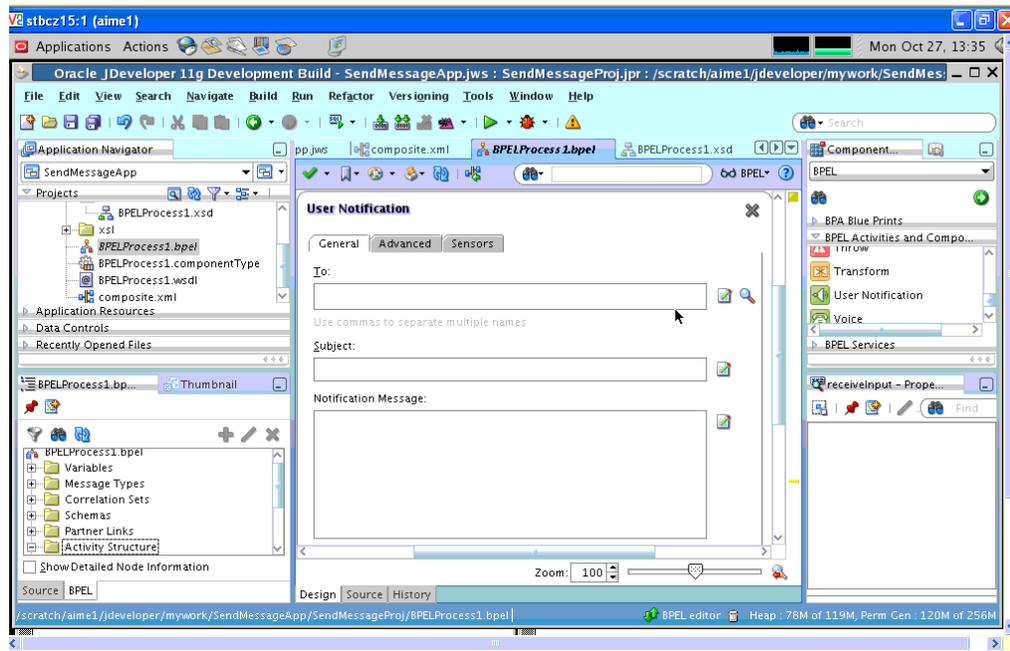
Figure I-8 Viewing the Expanded Process Element



14. To enable messaging in this process, drag and drop **User Notification** from **BPEL Activities and Components** located in the Component Palette between the **receiveInput** and **callbackClient** activities.

The User Notification activity appears (Figure I-9).

Figure I-9 User Notification Activity Before Configuring the Inputs

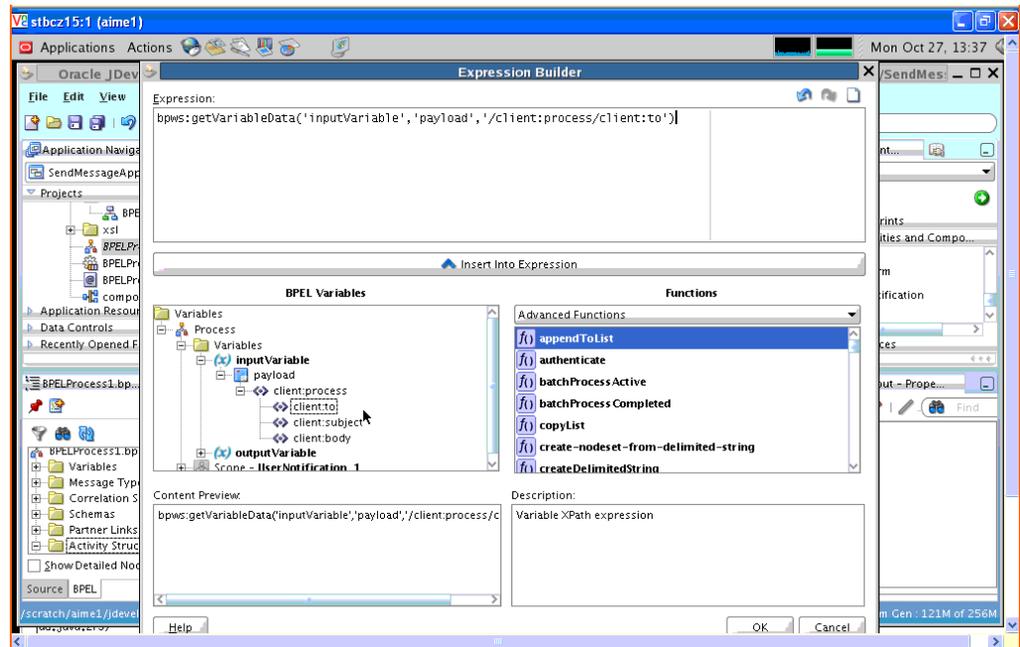


15. Click the XPath Expression Builder icon to the right of the **To:** input box.

16. Modify the expression for the **To** recipient, as follows:

- In the BPEL Variables pane, select **Variables**, **inputVariable**, **Payload**, **clientprocess**, and **client:to** (Figure I-10).
- Click **Insert Into Expression**.
- Click **OK**.

Figure I-10 Defining the Recipient ("to") Expression

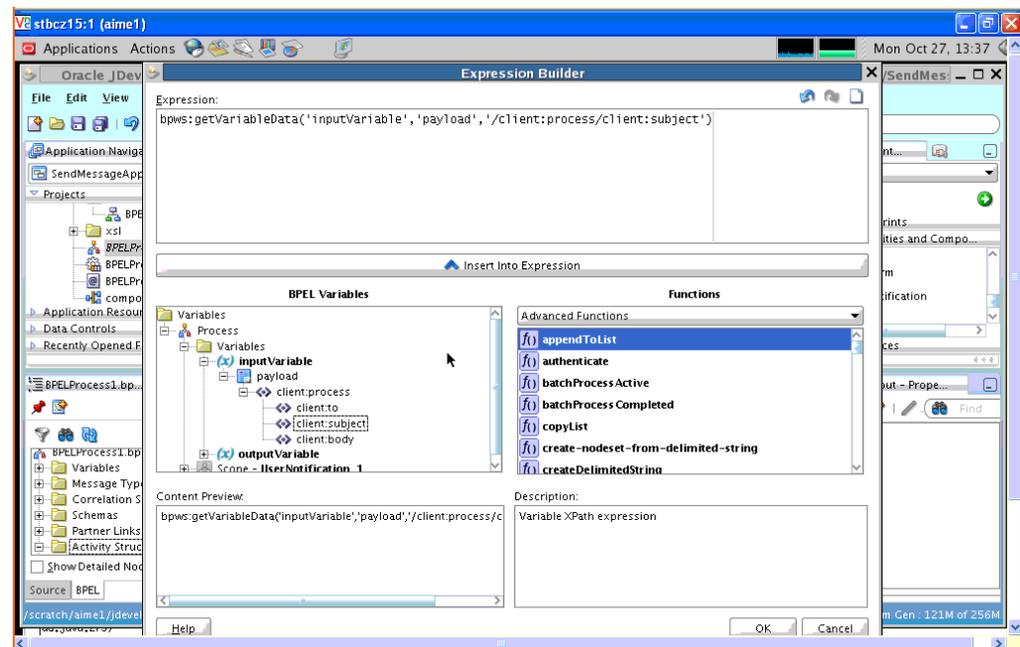


17. Click the XPath Expression Builder icon to the right of the **subject:** input box.

18. Modify the expression for the subject as follows:

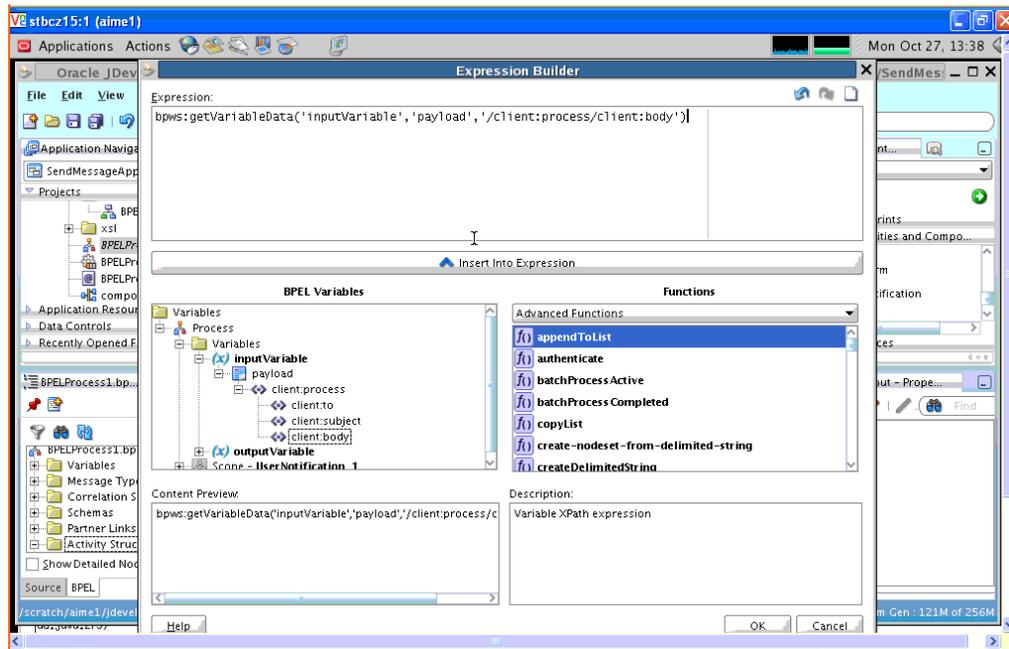
- In the BPEL Variables pane, select **Variables, InputVariable, Payload, clientprocess,** and **client:subject** (Figure I-11).
- Click **Insert Into Expression**.
- Click **OK**.

Figure I-11 Defining the Subject Expression



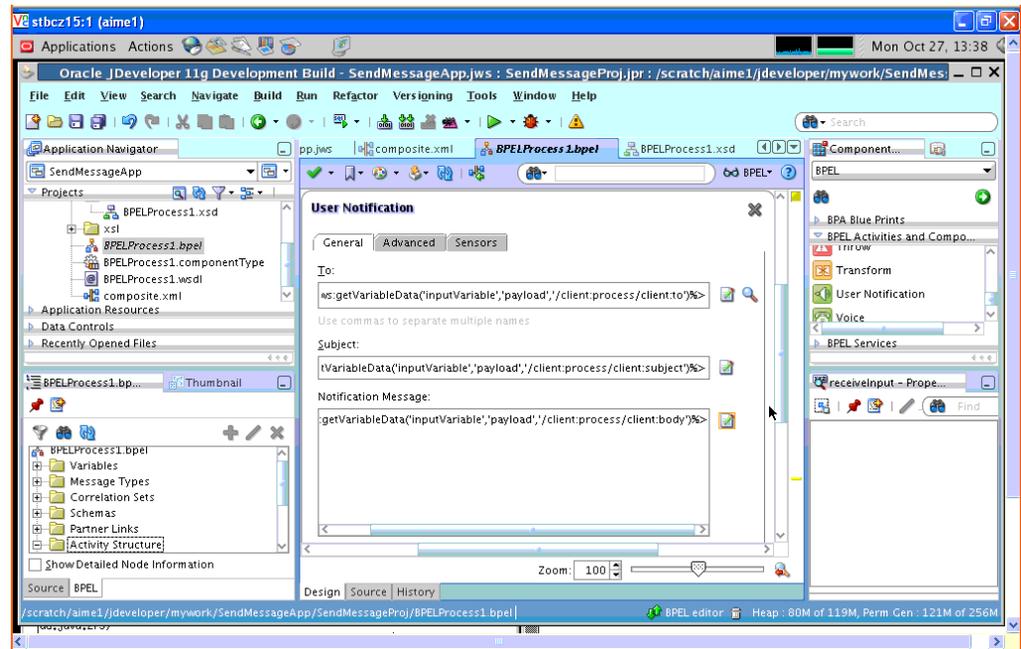
19. Click the XPath Expression Builder icon to the right of the **body:** input box.
20. Modify the expression for the body as follows:
 - In the BPEL Variables pane, select **Variables, InputVariable, Payload, client:process,** and **client:body** (Figure I-12).
 - Click **Insert Into Expression**.

Figure I-12 Defining the Body Expression



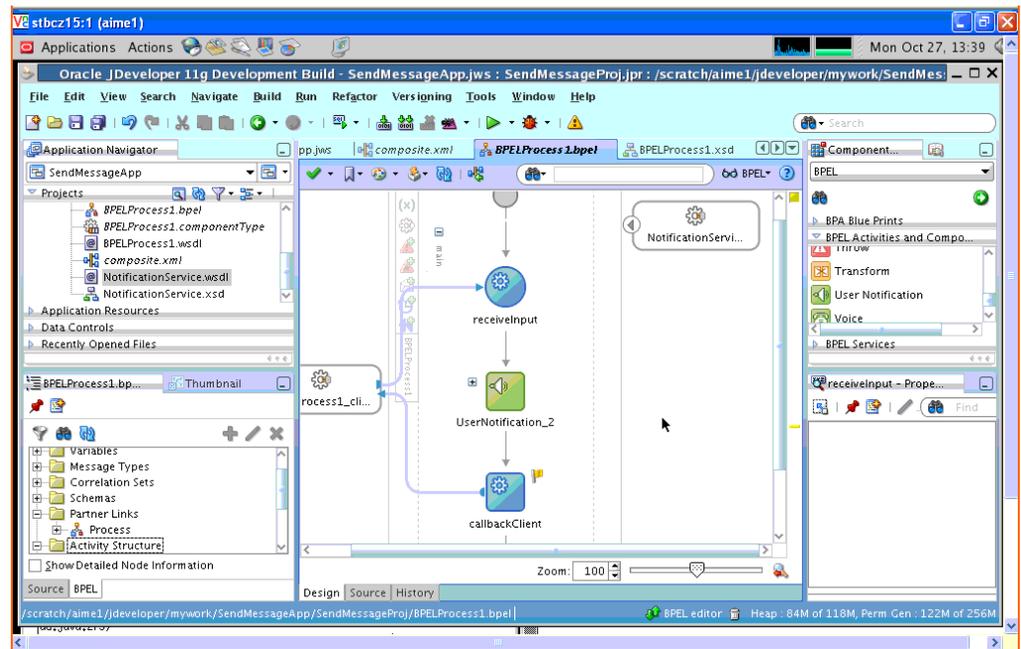
- Click **OK**.
- Click **Apply** and then **OK** to apply the changes (Figure I-13).

Figure I-13 Confirming the Changes to the Inputs



The changes to the inputs are saved and the configuration of the User Notification activity is complete. You can now see the **User Notification** activity in the BPEL application (Figure I-14). The SOA Composite is complete.

Figure I-14 User Notification Activity After Configuration of Inputs

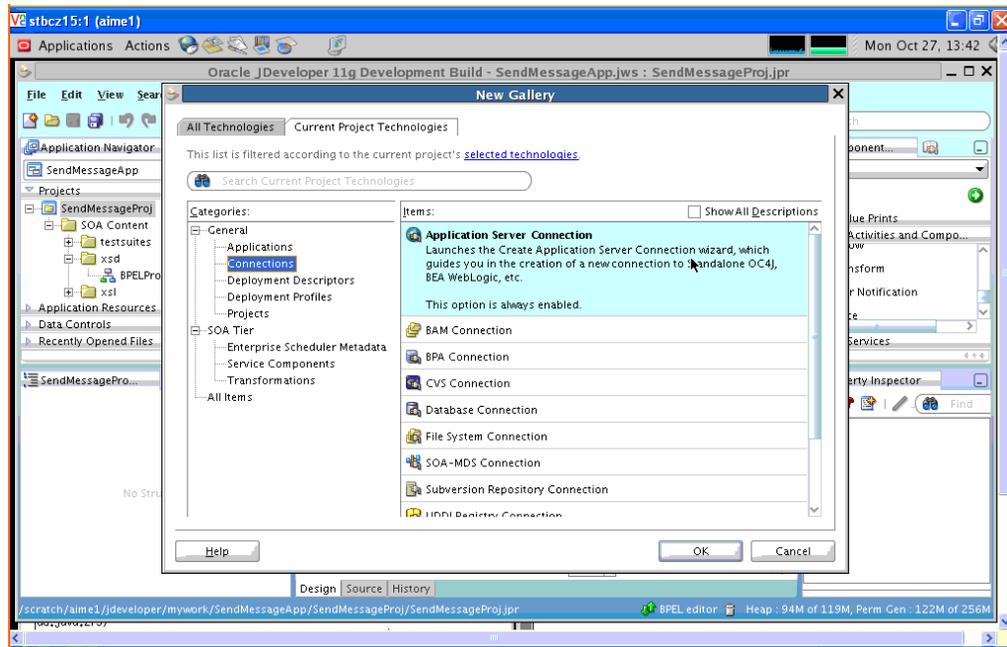


I.1.4 Creating a New Application Server Connection

Perform the following steps to create a new Application Server Connection.

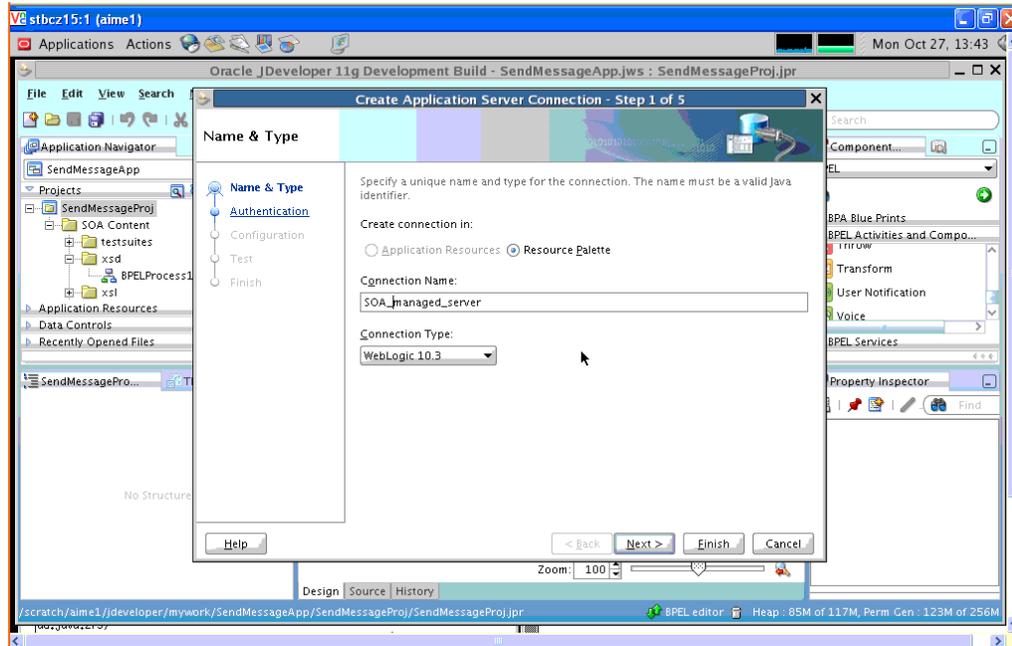
1. Create a new Application Server Connection by right-clicking the project and selecting **New, Connections, and Application Server Connection** (Figure I-15).

Figure I-15 New Application Server Connection



2. Name the connection `SOA_server` and click **Next** (Figure I-16).
3. Select **WebLogic 10.3** as the **Connection Type**.

Figure I-16 New Application Server Connection



4. Enter the authentication information. The typical value for username is `weblogic`.

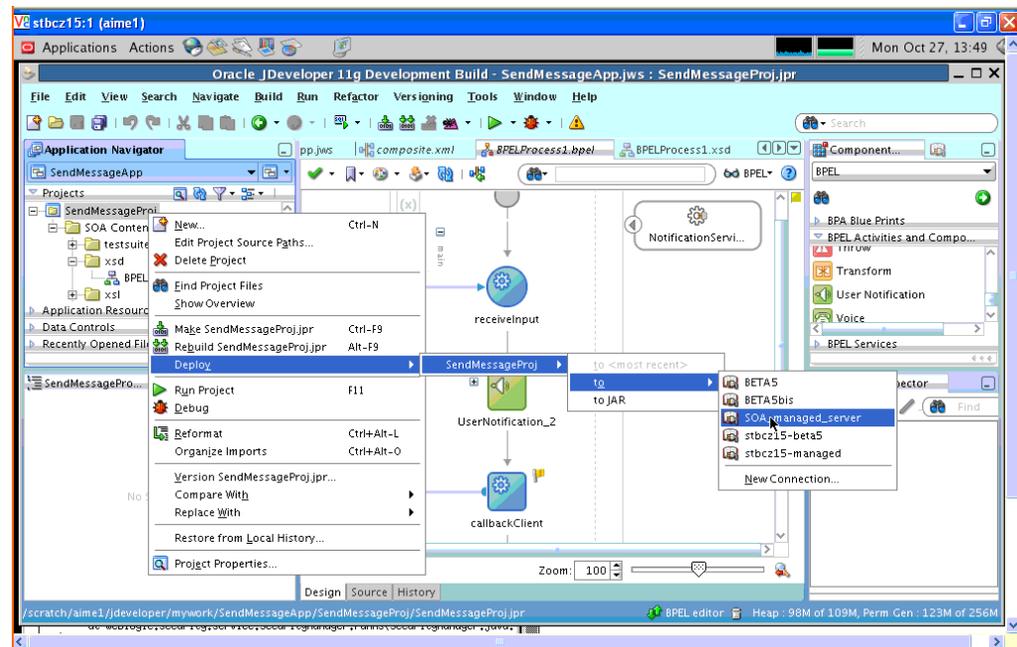
5. In the Connection dialog, enter the hostname, port and SSL port for the SOA admin server, and enter the name of the domain for WLS Domain.
6. Click **Next**.
7. In the Test dialog, click **Test Connection**.
8. Verify that the message **Success!** appears.
The application server connection has been created.

I.1.5 Deploying the Project

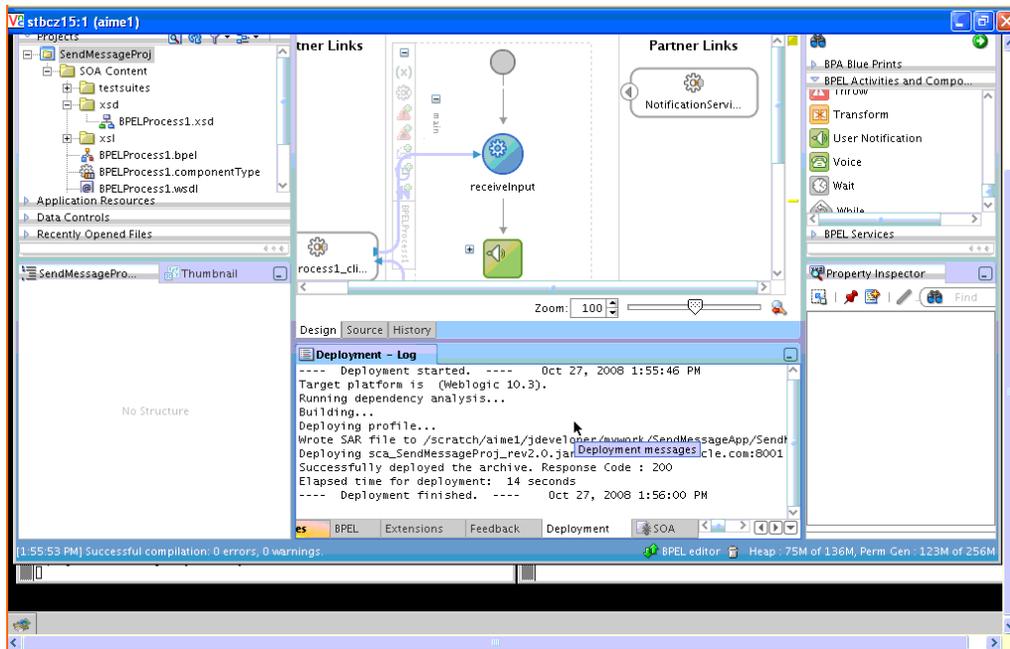
Perform the following steps to deploy the project:

1. Deploy the project by selecting the **SendMessage** project, **Deploy**, **SendMessageProj**, **to**, and **SOA_managed_server** (Figure I-17).

Figure I-17 Deploying the Project



2. Verify that the message **Build Successful** appears in the log.
3. Enter the default revision and click **OK**.
4. Verify that the message **Deployment Finished** appears in the deployment log (Figure I-18).

Figure I-18 Verifying that the Deployment is Successful

You have successfully deployed the application.

Before you can run the sample you must configure any additional drivers in Oracle User Messaging Service and configure a default device for the user receiving the message in User Messaging Preferences, as described in the following sections.

Note: Refer to *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite* for more information.

I.1.6 Configuring User Messaging Preferences

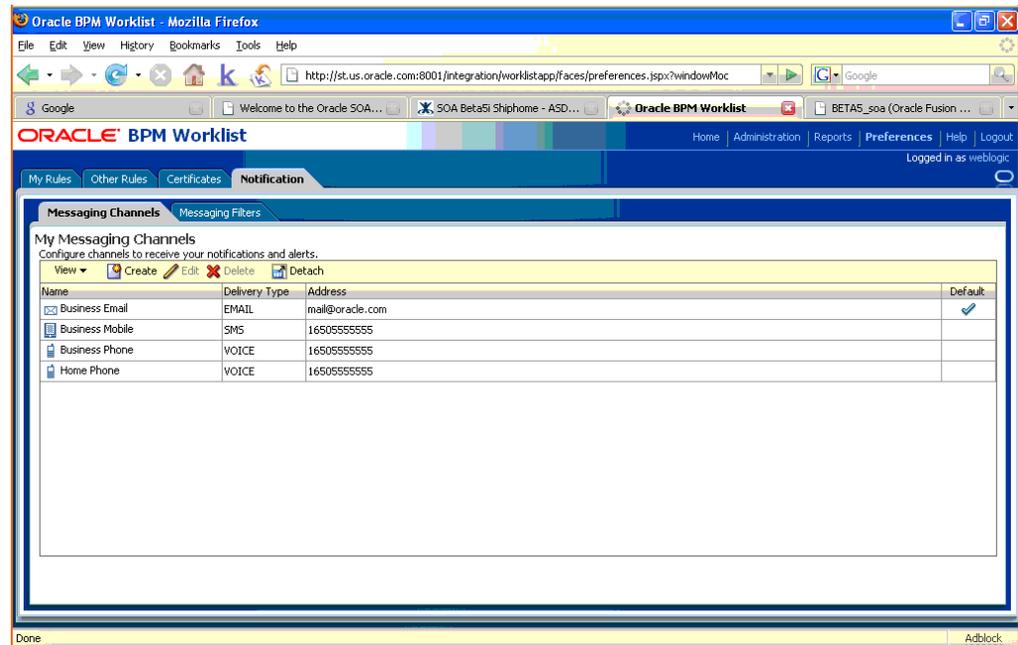
For users to receive the notifications, they must register the devices that they use to access messages through User Messaging Preferences. Perform the following steps:

1. Log in to the User Messaging Preferences application at one of the following URLs:
 - Directly at `http://server:port/sdpmessaging/userprefs-ui`
 - Through the Worklist application's Preferences > Notification tab at: `http://server:port/integration/worklistapp`

The User Messaging Preferences application appears.

2. Click the **Messaging Channels** tab (Figure I-19).

Figure I-19 Messaging Channels Tab



You are prompted for login credentials.

3. In the **Messaging Channels** tab, select a channel.
4. Set a channel as the default by expanding the device folder, and then clicking **Set as Default** adjacent to the selected channel.

A checkmark appears next to the selected channel, designating it as the default means of receiving notifications. All messages sent to that user will be sent to that channel.

I.1.7 Testing the Sample

The following steps describe how to perform a test message transmission through Enterprise Manager.

Perform the following steps to run and test the sample:

1. Open a web browser window and login to Enterprise Manager for the SOA domain. For example, `http://host:port/em`.
2. In Oracle Enterprise Manager, expand the SOA folder in the navigation tree, and click the deployed **SendMessageProj** composite application. Click the **Test** button to launch the test client page.
3. In the **Input Arguments** section provide the input values for invoking **SendMessageProj**.

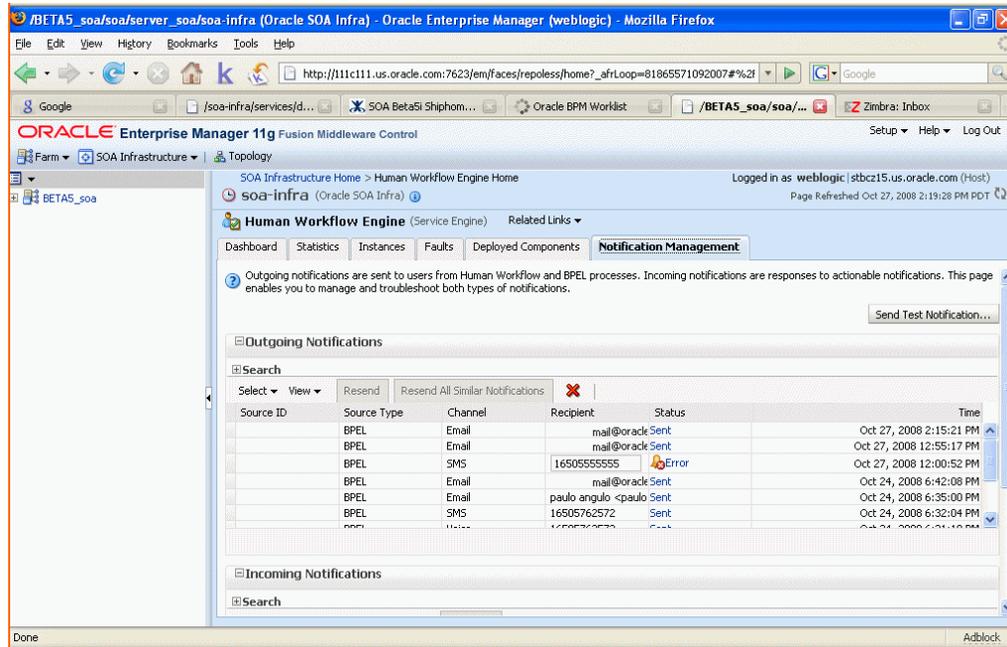
Enter the following values:

- **to:** weblogic (the user)
 - **subject:** notification test (the subject)
 - **body:** the message content
4. Click **Test Web Service**.

I.1.7.1 Verifying the Execution of Sending the Email

Log in to the Human Workflow Engine. Verify the outgoing notifications and their statuses from the Notification Manager tab. (Figure I-20).

Figure I-20 Viewing Outgoing Notifications



I.2 Send Email with Attachments

This section describes how to build and run the Send Email with Attachments application provided with Oracle User Messaging Service.

Note: To learn about the architecture and components of Oracle User Messaging Service, see *Oracle Fusion Middleware Getting Started with Oracle SOA Suite*.

This chapter contains the following sections:

- Section I.2.1, "Overview"
- Section I.2.2, "Installing and Configuring SOA and User Messaging Service"
- Section I.2.3, "Running the Pre-Built Sample"
- Section I.2.4, "Testing the Sample"
- Section I.2.5, "Building the Sample"
- Section I.2.6, "Creating a New Application Server Connection"

I.2.1 Overview

The "Send Email With Attachment" application demonstrates a BPEL process that sends an email with an attached file.

A BPEL process looks up a user's email address from the identity store, reads a file from the file system, creates email content and then sends an email to the user. [Section I.2.5, "Building the Sample"](#) shows you how to add an email with attachments to your SOA composite application, allowing your applications to be enabled with messaging. If you want to model the application from scratch, go to the section titled Building the Sample. Or, you can directly use the pre-built project provided with this tutorial.

Before you run the pre-built sample or build the application from scratch, you must install and configure the server as described in [Section I.2.2, "Installing and Configuring SOA and User Messaging Service"](#). By default, soa-infra does not send out notifications. The following steps describe installing and configuring the email drivers needed to communicate with the email server.

I.2.1.1 Provided Files

The following files are included in the sample application:

- ns_sendemail.pdf – this document.
- Project – the directory containing Oracle JDeveloper project files.
- Readme.txt.
- Release notes

I.2.2 Installing and Configuring SOA and User Messaging Service

The installation of SOA and User Messaging Service has already been performed on your hosted instance, and the sample user, `weblogic`, has already been created. Perform the following steps to enable notifications in soa-infra, if not already done:

1. Using Enterprise Manager, go to the **SOA Infrastructure** menu, and select **SOA Administration > Workflow Notification Properties**, and set **Notification Mode** to **ALL**.
2. Configure the User Messaging drivers if required as described in "Configuring Drivers" in the *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite*.
3. Set the email address for user `weblogic` by using the JXplorer LDAP browser. Refer to ["Updating Addresses in Your LDAP User Profile"](#).
4. Restart the server.

I.2.2.1 Updating Addresses in Your LDAP User Profile

Perform the following steps to set the email address for user `weblogic` by using the JXplorer LDAP browser:

I.2.2.1.1 Installing Download and install JXplorer from <http://www.jxplorer.org>.

I.2.2.1.2 Connecting 1. Set the embedded LDAP server admin password as follows:

- Login to the Oracle WebLogic Server Administration Console.
 - Click the domain name link > **Security > Embedded LDAP**.
 - Enter a new Credential and Confirm Credential (for example, `weblogic`).
 - Click **Save**.
2. Connect from JXplorer by specifying the fields in [Table I-2](#):

Table I–2 JXplorer Connection Fields

Field	Value
Host	WLS AdminServer hostname
Port	WLS AdminServer port
Protocol	LDAP v3
Security Level	User + Password
User DN	cn=Admin
Password	password

I.2.2.1.3 Setting User Messaging Device Addresses in LDAP The following example uses the user `weblogic`. You may create and use additional users.

1. Expand the LDAP tree as follows: **domain > myrealm > people > weblogic**.
2. Click the user entry.
3. Select the HTML view tab on the right.
4. Enter the necessary Email Address and Mobile Phone Number.
5. Click **Submit**.

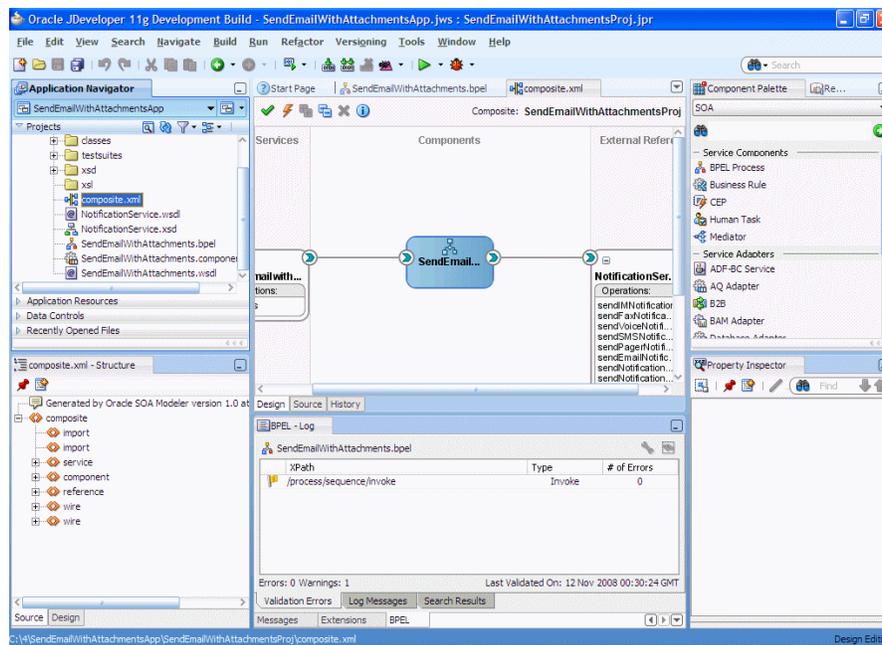
I.2.3 Running the Pre-Built Sample

Perform the following steps to run and deploy the prebuilt sample application:

1. Open **SendEmailWithAttachmentsApp.jws** (contained in the .zip file) in Oracle JDeveloper.

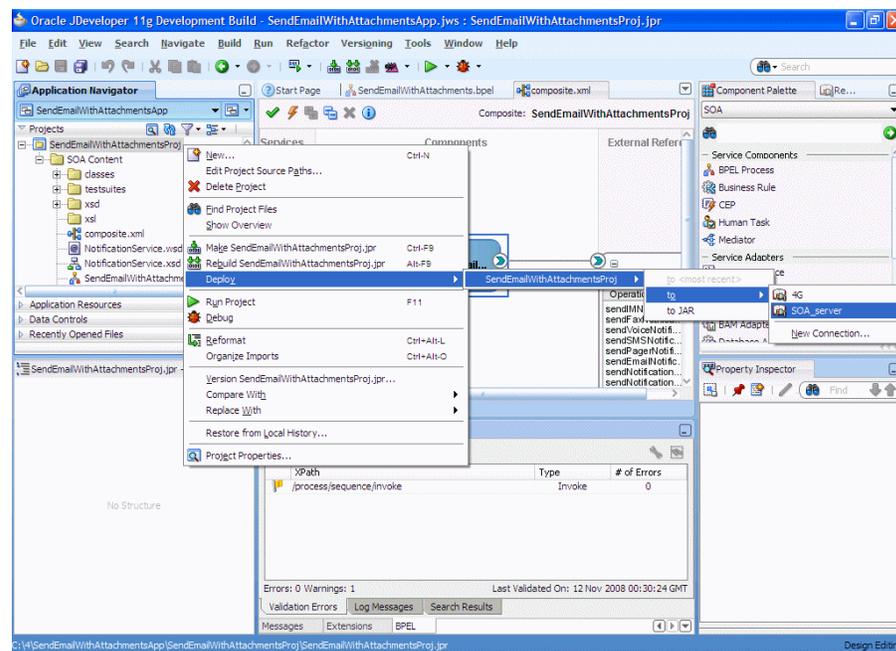
In the Oracle JDeveloper main window you can view the following components of the sample application under the **Composite.xml** tab.

Figure I–21 Oracle JDeveloper Main Window



- The left box is the definition of a web service client that is used to initiate an application.
 - The middle box is a BPEL process that creates and formats the message and calls the messaging service.
 - The right box is the messaging service resource that is used to send the message.
2. Create an Application Server Connection by right-clicking the project in the navigation pane and selecting New. Follow the instructions in [Section I.2.6, "Creating a New Application Server Connection."](#)
 3. Deploy the project by selecting the **SendEmail** project, **Deploy**, **SendEmailProj**, **to**, and **SOA_server** ([Figure I-22](#)).

Figure I-22 Deploying the Project



4. Verify that the message **Build Successful** appears in the log.
5. Enter the default revision and click **OK**.
6. Verify that the message **Deployment Finished** appears in the deployment log.

You have successfully deployed the application.

Before you can run the sample you must configure any additional drivers in Oracle User Messaging Service and configure a default device for the user receiving the message in User Messaging Preferences, as described in the following sections.

Note: Refer to *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite* for more information.

I.2.4 Testing the Sample

The following steps describe how to perform a test message transmission through Enterprise Manager.

Perform the following steps to run and test the sample:

1. Open a web browser window and login to Enterprise Manager for the SOA domain. For example, `http://host:port/em`.
2. In Enterprise Manager, expand the SOA folder in the navigation tree, and click the deployed **SendEmailWithAttachmentsProj** composite application. Click the **Test** button to launch the test client page.
3. In the **Input Arguments** section provide the input values for invoking **SendEmailWithAttachmentsProj**.

Enter the following values:

- **to:** weblogic (the user)
- **subject:** notification test (the subject)
- **body:** the message content
- **attachmentName:** the name of the being attached, including extension.
- **attachmentMimeType:** for example, `image/gif`.

To send files such as PDF, DOC, GIF, or JPEG files, the following values can be used for the `attachmentMimeType` entry:

- `file-name.doc` – `attachmentMimeType: application/msword`
 - `file-name.pdf` – `attachmentMimeType: application/pdf`
 - `file-name.jpg` – `attachmentMimeType: image/jpeg`
 - `file-name.gif` – `attachmentMimeType: image/gif`
 - **attachmentURI:** the URI for the attachment
4. Click **Test Web Service**.

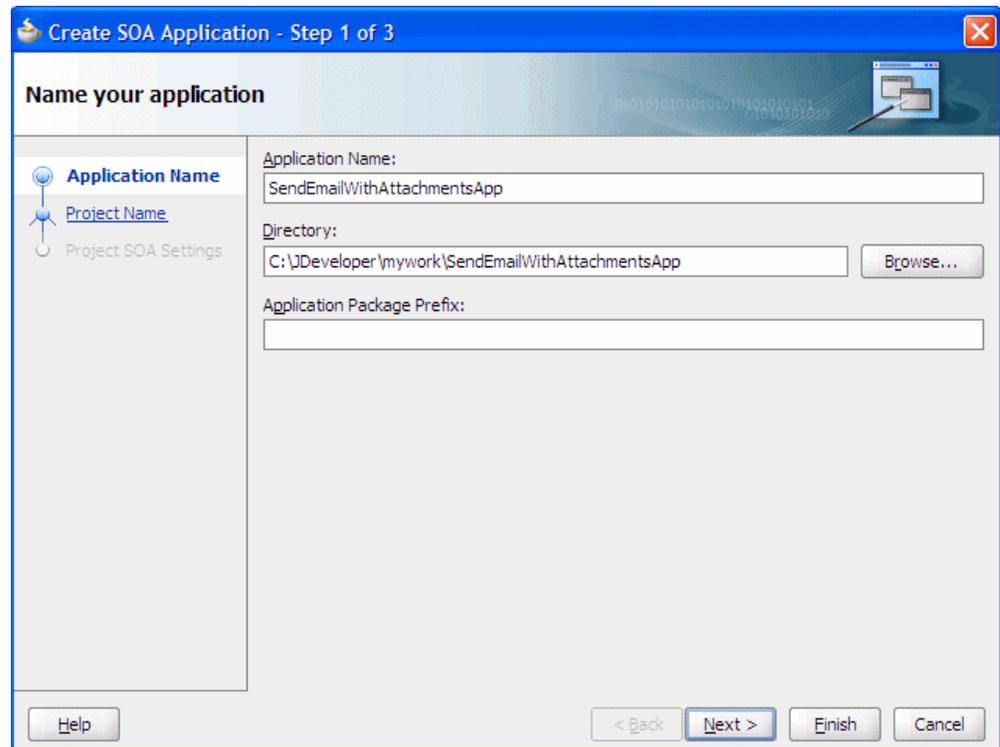
I.2.4.1 Verifying the Execution

Check the weblogic email account. It should have received an email with attachment.

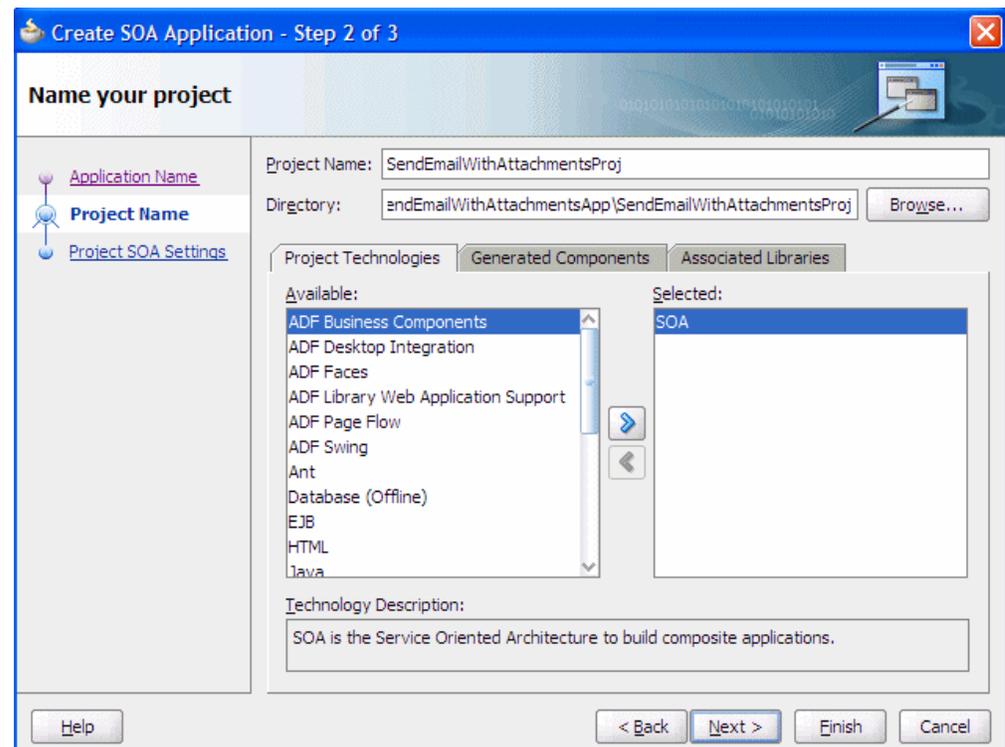
I.2.5 Building the Sample

Performing the following procedure of building the sample from scratch allows you to learn how to add messaging to your SOA Composite Applications, and use User Messaging Preferences.

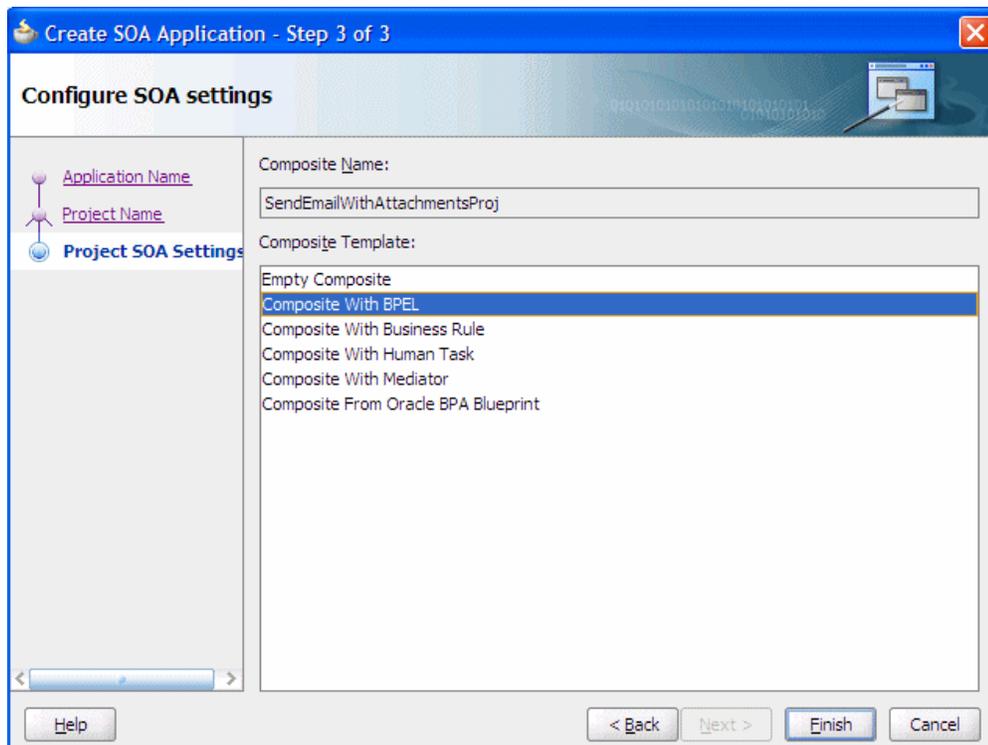
1. Open Oracle JDeveloper 11g.
2. Create a new application by selecting **File, New, Applications, and SOA Application**. Click **OK**.
3. Enter the *Application Name* and click **Next** (Figure I-23).

Figure I-23 Creating a New Application and Project (1 of 3)

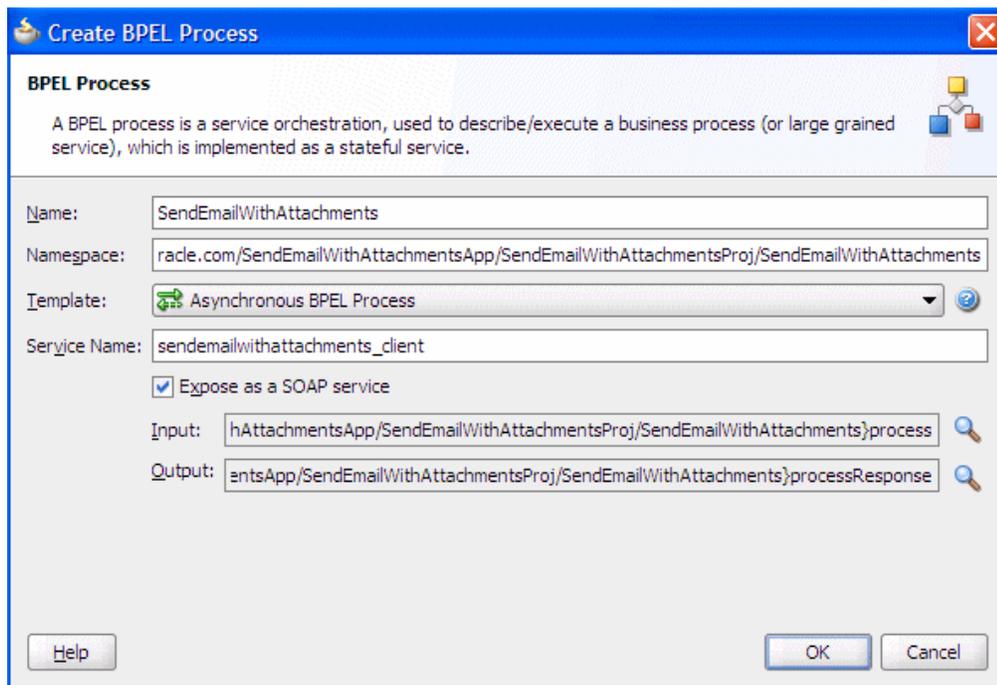
4. Enter the name for the project and click **Next** (Figure I-24).

Figure I-24 Creating a New Application and Project (2 of 3)

5. Select the **Composite With BPEL** composite template (Figure I-25). Click **Finish**.

Figure I-25 Creating a New Application and Project (3 of 3)

6. In the Create BPEL Process dialog, enter the BPEL process name as `SendEmailWithAttachments` (Figure I-26). Click **OK**.

Figure I-26 Creating the BPEL Process

7. Verify that **Expose as a SOAP service** is checked. Click **OK**.

8. You have now created an empty and default BPEL application.

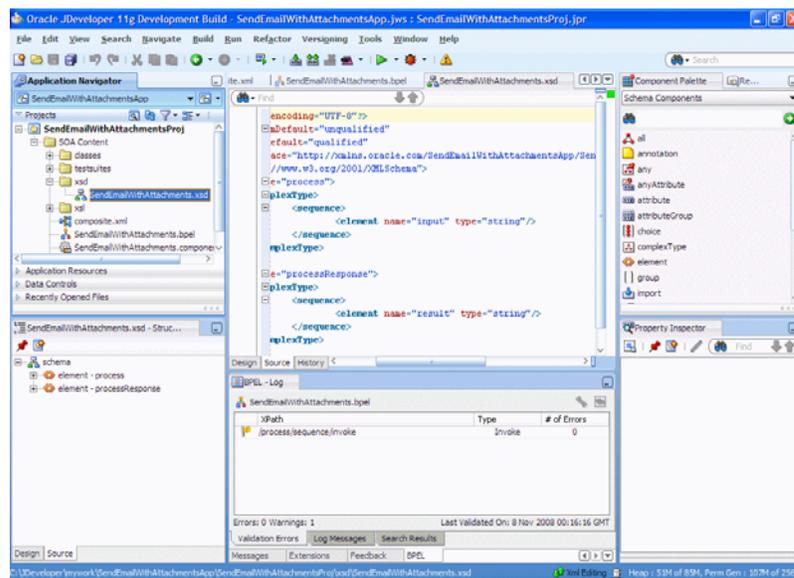
In the Oracle JDeveloper main window you can view the following components of the sample application under the *Composite.xml* tab.

- The left box is the definition of a web service client that is used to initiate an application.
- The middle box is a BPEL process that creates and formats the message and calls the messaging service.

Note: You will later create the messaging service resource that is used to send the message when you create the User Notification BPEL process (steps 13-19).

9. Expand the *xsd* folder in the Application Navigator and open *SendEmailWithAttachments.xsd* by double-clicking it (Figure I-27).

Figure I-27 Accessing the SendEmailWithAttachments.xsd File



10. Click the **Source** tab (Figure I-27).

11. Perform the following modifications to the inputs of this BPEL application:

In the generated file, *SendEmailWithAttachments.xsd*, in the *xsd* folder in the Application Navigator under projects, the following element definition is created by default:

```
<element name="process">
  <complexType>
    <sequence>
      <element name="input" type="string"/>
    </sequence>
  </complexType>
</element>
```

Select the **Source** tab, and replace the lines above with the following:

```
<element name="process">
```

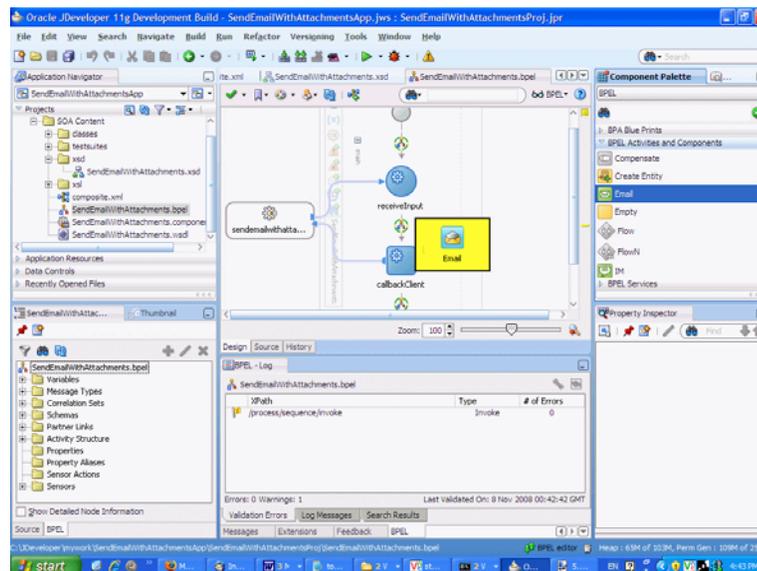
```

<complexType>
  <sequence>
    <element name="to" type="string"/>
    <element name="subject" type="string"/>
    <element name="body" type="string"/>
    <element name="attachmentName" type="string"/>
    <element name="attachmentMimeType" type="string"/>
    <element name="attachmentURI" type="string"/>
  </sequence>
</complexType>
</element>

```

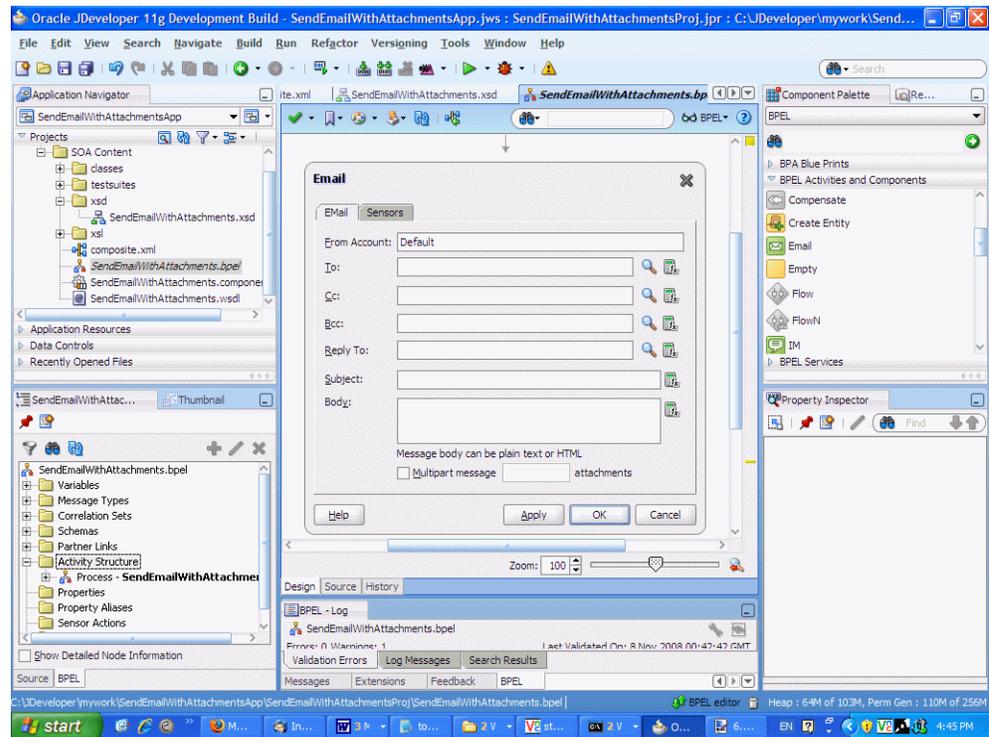
This xsd element defines the input for the BPEL process.

Figure I-28 Editing Email



12. Save the project.
13. Select the **SendEmailWithAttachments.bpel** editor screen.
14. Drag and drop an **Email** activity from **BPEL Activities and Components** located in the **Component Palette** between the **receiveInput** and **callbackClient** activities (Figure I-28).
15. In the Edit Email window, leave the From account as **Default**.

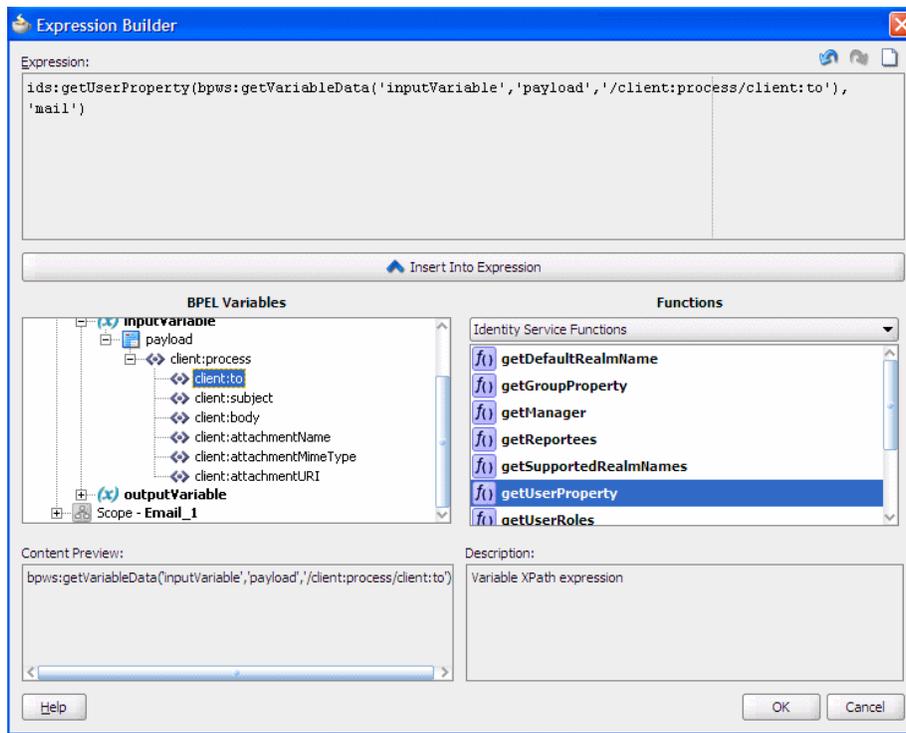
Figure I-29 Edit Email Window



16. To create the expression for **To**, select the Expression Builder (the second icon, [Figure I-30](#)) and perform the following steps:
- Select **Identity Service Functions** from the functions dropdown list.
 - Select the **getUserProperty()** function and select **Insert into Expression**.
 - Under **BPEL variables** select **Variables > Process > Variables >inputVariable > payload > client:process > client:to**.
 - Click **Insert into Expression**.
 - Type the string `mail` manually.
 - Correct the parenthesis so they are matched.
 - Click **OK**.

This expression ([Figure I-30](#)) takes the data from the web service and maps it to the business email of the local SOA user.

Figure I-30 Expression Builder for the To Path

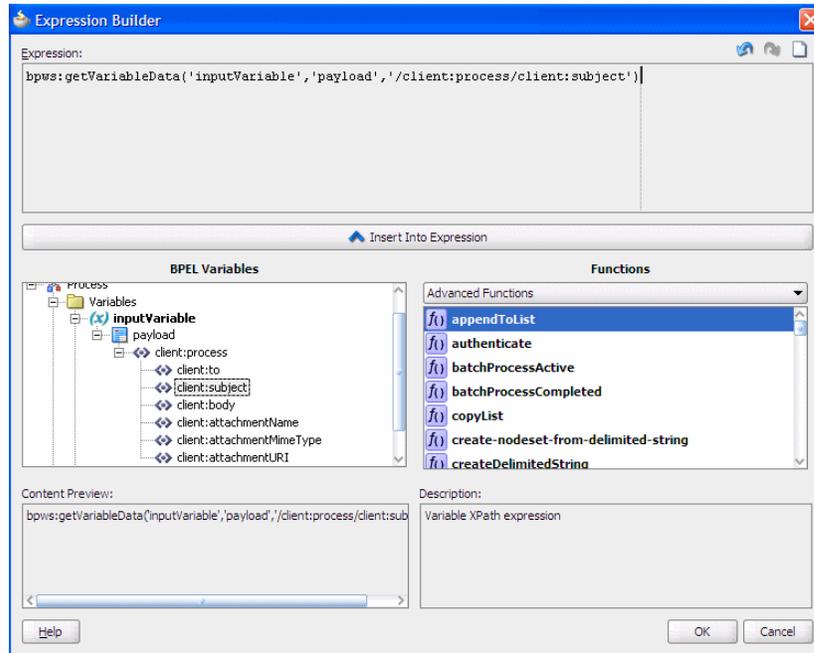


The expression should appear as follows:

```
ids:getUserProperty( bpws:getVariableData('inputVariable',
'payload', '/client:process/client:to'),
'mail')
```

- For **Subject**, select the Expression builder. Select **getVariableData** from **Functions** and click **Insert Into Expression**.

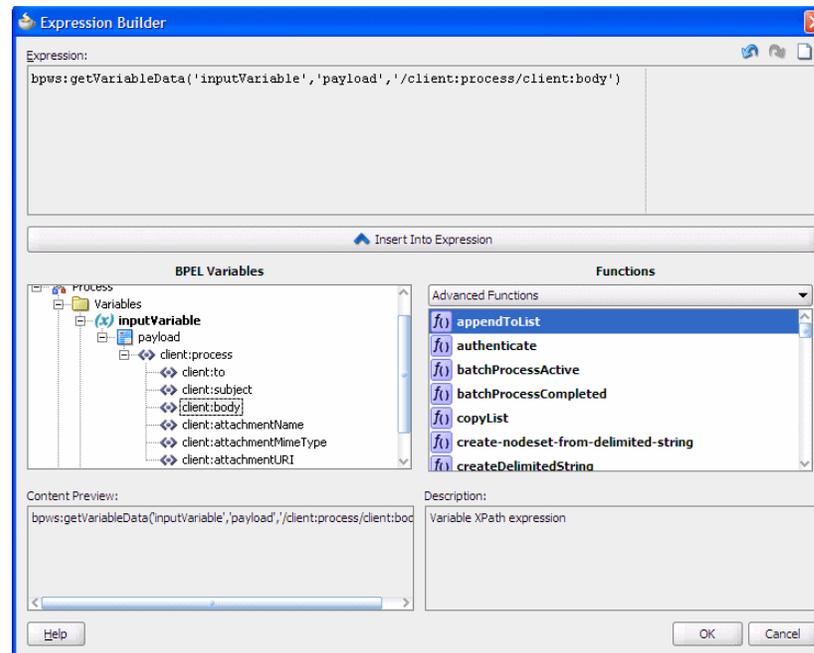
Figure I-31 shows the Expression Builder for the **Subject**.

Figure I-31 Expression Builder for the Subject

The expression should appear as follows:

```
bpws:getVariableData( 'inputVariable', 'payload',
'/client:process/
client:subject')
```

18. For **Body**, select the Expression Builder and set the expression as shown in [Figure I-32](#).

Figure I-32 Expression Builder for the Body

The expression should appear as follows:

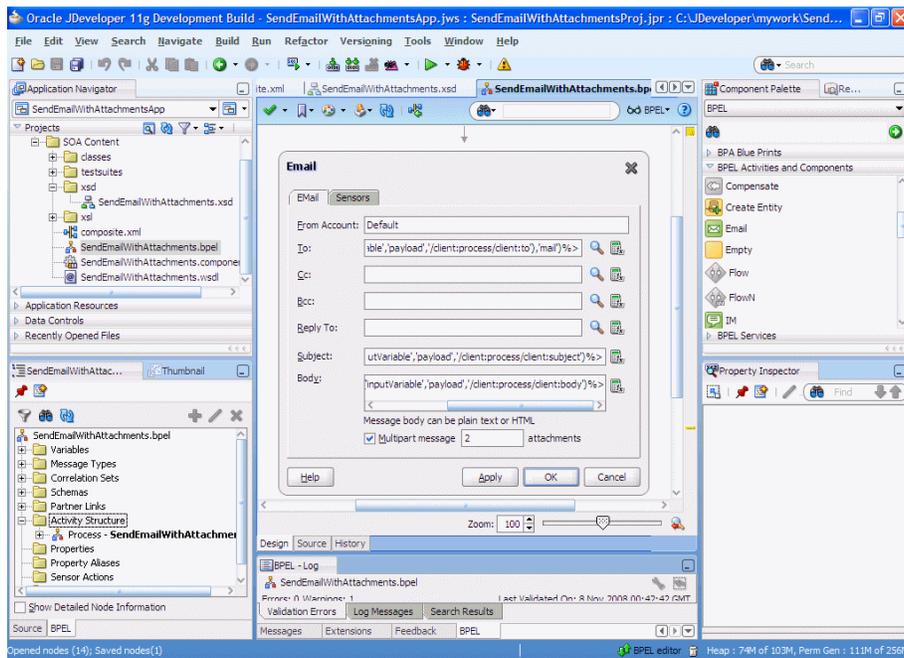
```
bpws:getVariableData('inputVariable','payload','/client:process/client:body')
```

19. In the Edit Email dialog (Figure I-33), ensure that the Multipart Message with attachments box is checked.

When an email has multiple parts, the attachment count includes the body that is set with the Wizard above. The body specified by the Wizard above is set as the first body part.

For example, to represent a multipart mail with one (1) attached file, enter 2 as the number of body parts. When there is one attachment, enter 1 as the number of body parts.

Figure I-33 Edit Email Window



20. Set the attachments:

Each body part has three attributes: **MimeType**, **BodyPartName**, and **ContentBody**. By default, the wizard generates default names, MIME types and contents for each of the attachments.

The assignment of these body parts has to be changed to set the correct data by modifying the copy rules in the assign activity in the notification scope. The copy rules (specified in the **Copy Operation** tab) are grouped for each assignment in the following order (the `copy-to` constructs are also listed):

```
MimeType - <to variable="varNotificationReq" part="EmailPayload"
```

```
query="/EmailPayload/ns1:Content/ns1:ContentBody/ns1:MultiPart/ns1:BodyPart[2]/ns1:MimeType"/>
```

```
Name - <to variable="varNotificationReq" part="EmailPayload"
```

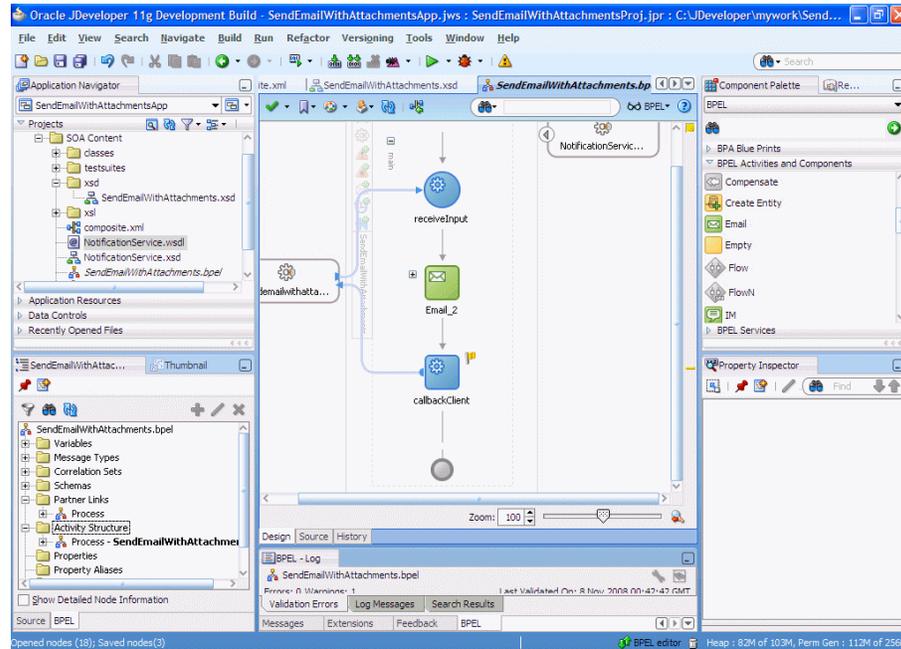
```
query="/EmailPayload/ns1:Content/ns1:ContentBody/ns1:MultiPart/ns1:BodyPart[2]/ns1:BodyPartName"/>
```

```
Contents - <to variable="varNotificationReq" part="EmailPayload"
```

```
query="/EmailPayload/ns1:Content/ns1:ContentBody/ns1:MultiPart/ns1:BodyPart[2]/ns1:ContentBody"/>
```

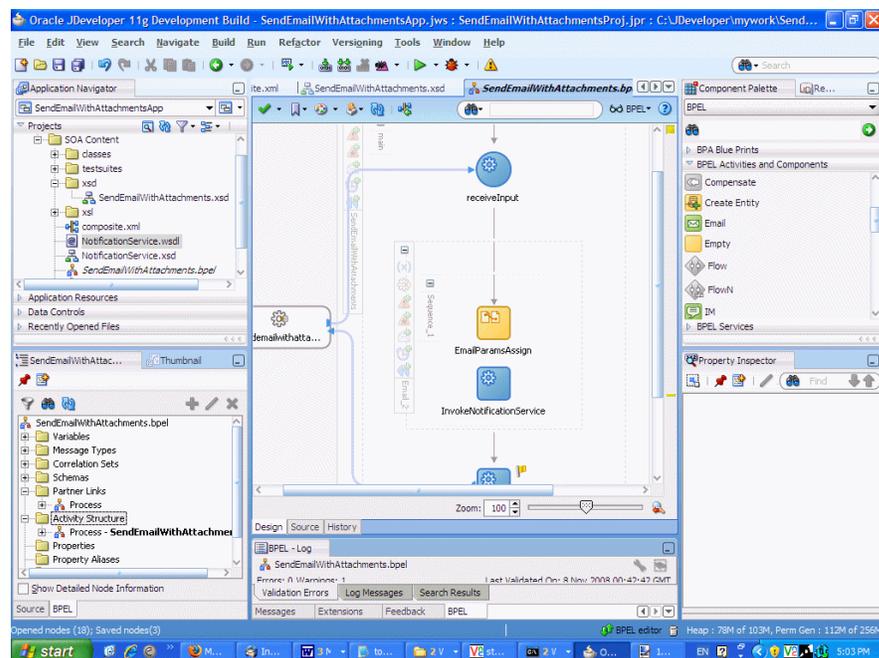
1. Expand the Email node by selecting the plus sign icon (Figure I-34).

Figure I-34 Expanding the Email Node



2. Double-click the EmailParamAssign node (Figure I-35).

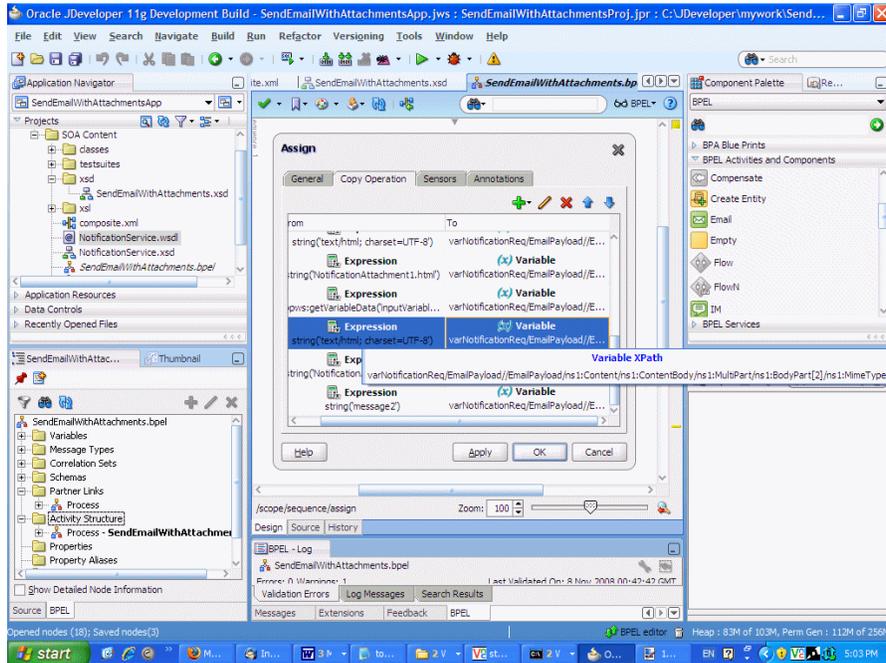
Figure I-35 Email ParamAssign Node



When making changes in the **EmailParamAssign** node (for example, editing the XPath variables), perform a **Save All** from the **File** menu after making each change. This will ensure that the changes are reflected in the .bpel file.

3. To edit the **contentType** of the second body part (the first body part is the contents set in the wizard) select the second body part variable ending with **MimeType** by double-clicking it ([Figure I-36](#)).

Figure I-36 Editing the *contentType* of the Second Body Part

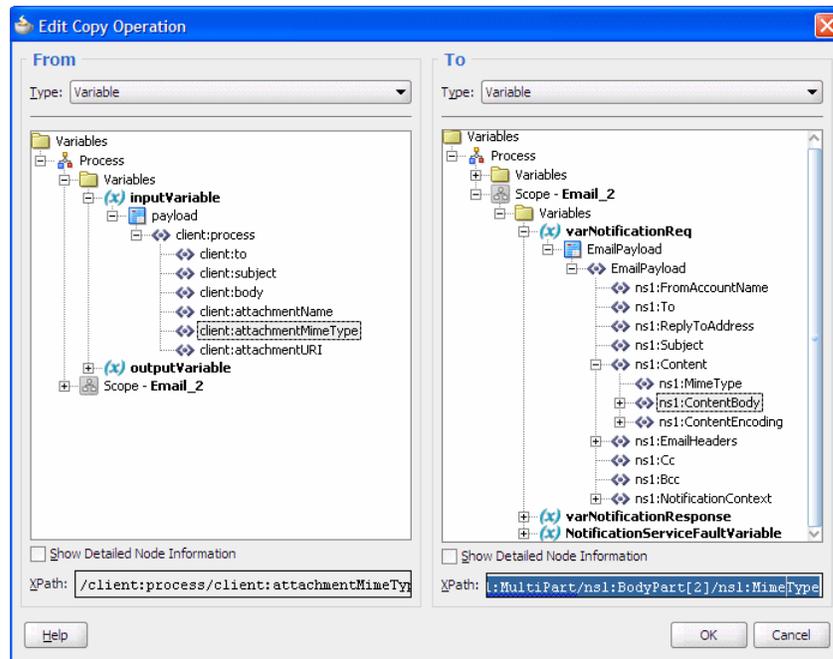


4. Edit the XPath as shown below ([Figure I-37](#)):

From: `/client:process/client:attachmentMimeType,`

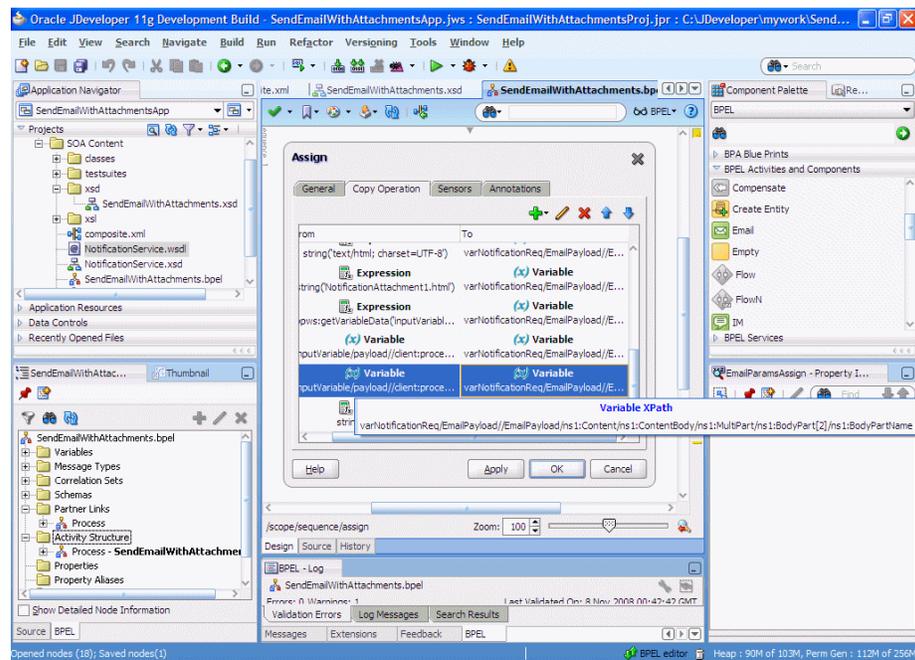
To: `/EmailPayload/ns1:Content/ns1:ContentBody/ns1:MultiPart/
ns1:BodyPart[2]/ns1:MimeType`

Figure I-37 Editing the XPath for mimeType



5. Save the project.
6. To edit the attachment name for the second attachment, select the second body part variable ending with **BodyPartName** by double-clicking it (Figure I-38).

Figure I-38 Editing the Attachment Name for the Second Attachment



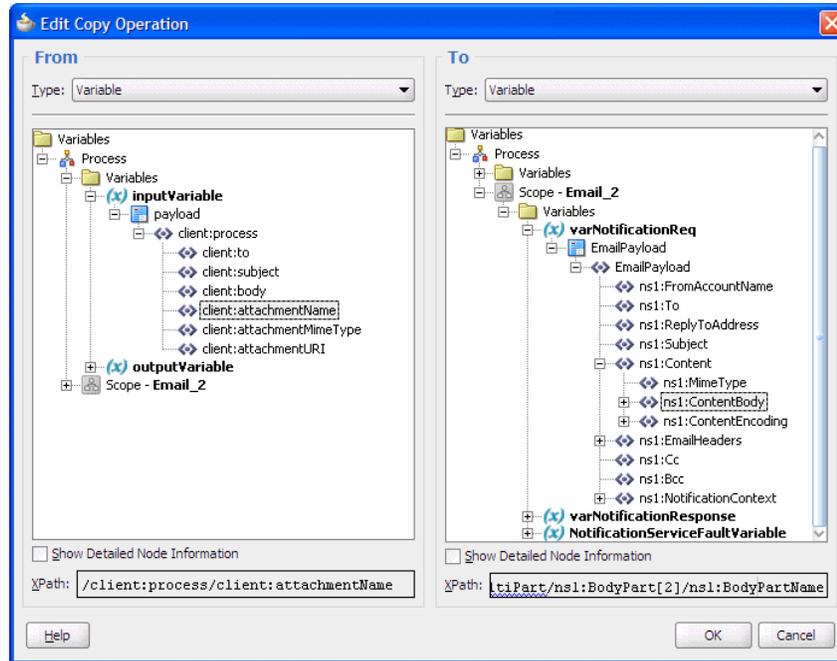
7. Edit the XPath as shown below:

From: /client:process/client:attachmentName

To: /EmailPayload/ns1:Content/ns1:ContentBody/ns1:MultiPart

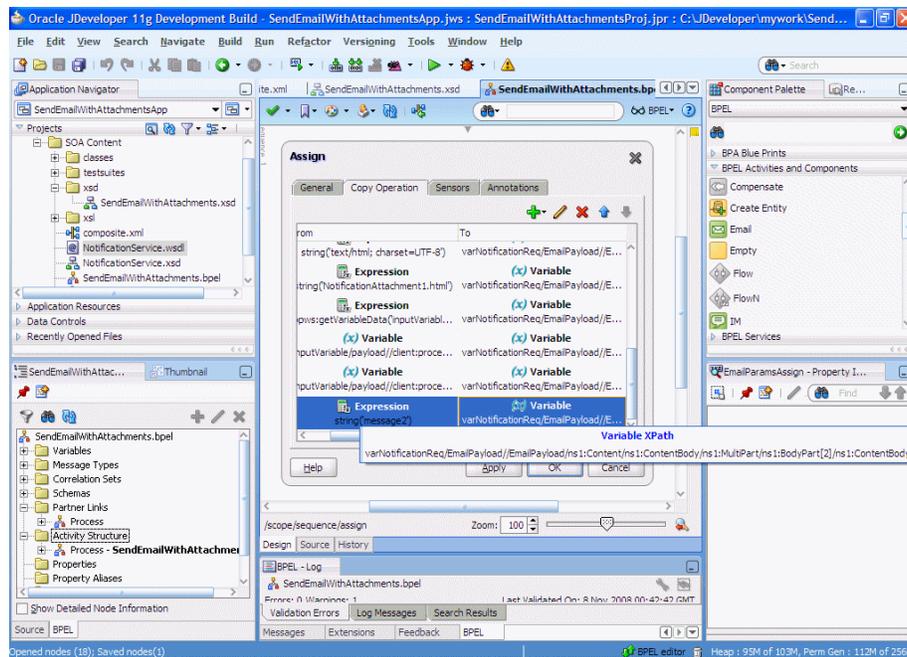
/ns1:BodyPart[2]/ns1:BodyPartName

Figure I-39 Editing the XPath for BodyPartName



8. Save the project.
9. To edit the attachment contents of the second attachment, select the second body part variable ending with **ContentBody** by double-clicking it (Figure I-40).

Figure I-40 Editing the Attachment Contents of the Second Attachment



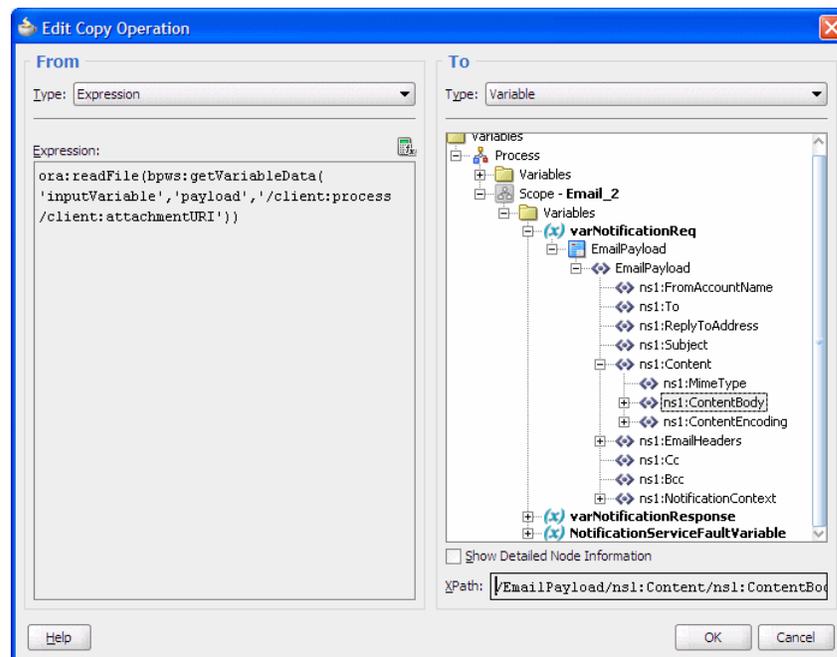
10. Edit the XPath as shown below (Figure I-41):

```
From: ora:readFile(bpws:getVariableData('inputVariable','payload','/client:
process/client:attachmentURI'))
```

```
To:
/EmailPayload/ns1:Content/ns1:ContentBody/ns1:MultiPart/
ns1:BodyPart[2]/ns1:ContentBody
```

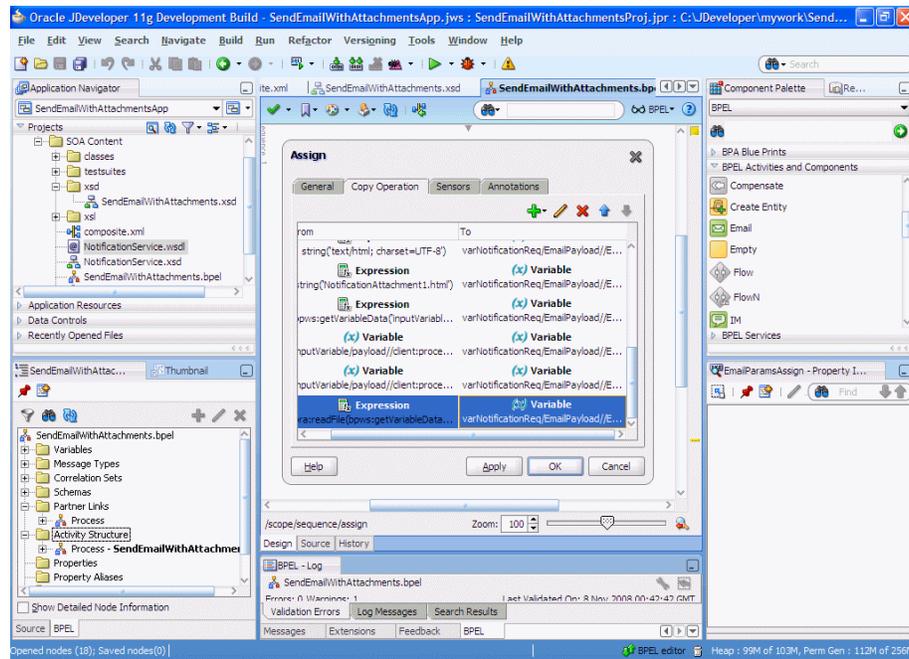
The `ora:readFile()` XPath function is available under **BPEL Xpath Extension Functions**.

Figure I-41 Editing the XPath from the ContentBody



11. Click OK in the Edit Copy Operation dialog.

Figure I-42 Copy Operations Tab



12. Click OK in the assign activity. Save the project.

The Process Modeling procedure is complete. You can use the information in this procedure to add notification with attachments to your SOA composite application.

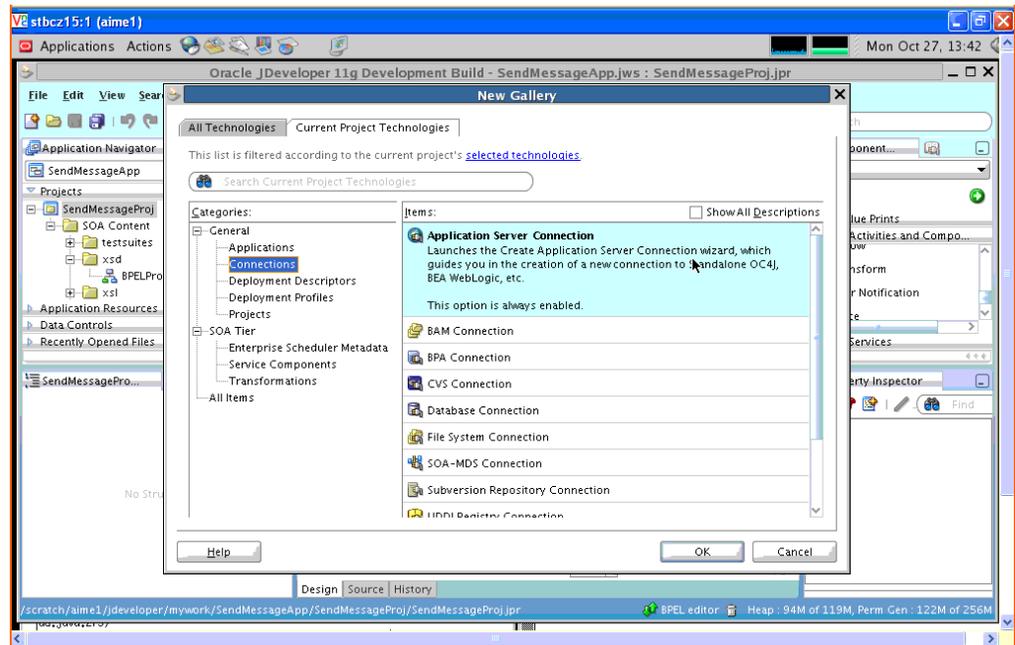
You can now deploy and run the application as described in [Section I.2.3, "Running the Pre-Built Sample."](#)

I.2.6 Creating a New Application Server Connection

Perform the following steps to create a new Application Server Connection.

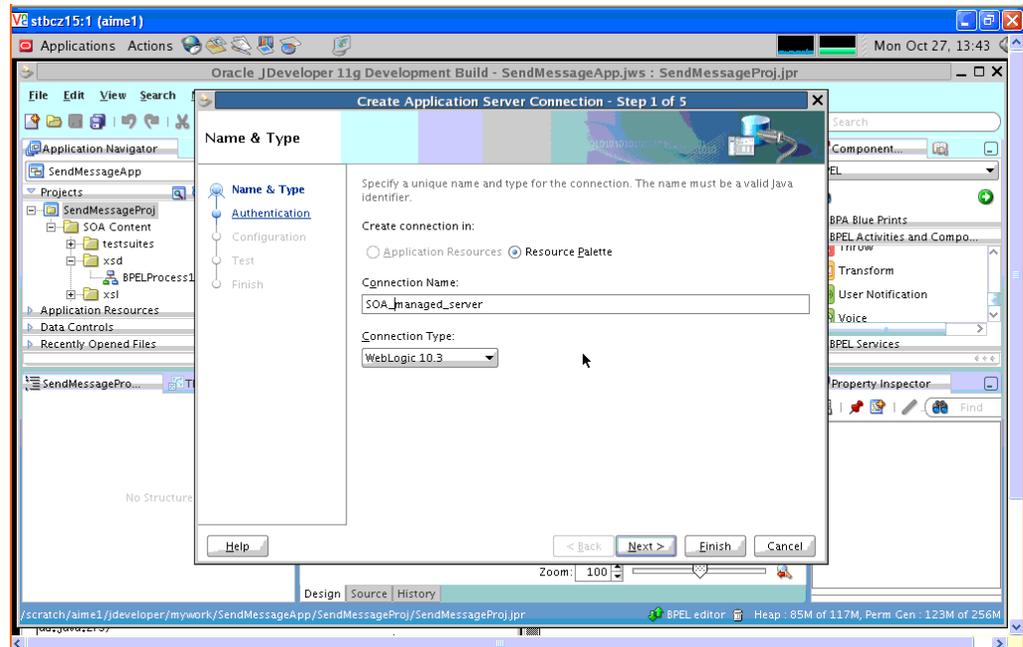
1. Create a new Application Server Connection by right-clicking the project and selecting **New, Connections, and Application Server Connection** (Figure I-43).

Figure I-43 New Application Server Connection



2. Name the connection SOA_server and click Next (Figure I-44).
3. Select WebLogic 10.3 as the Connection Type.

Figure I-44 New Application Server Connection



4. Enter the authentication information. The typical value for username is weblogic.
5. On the Connection dialog, enter the hostname, port and SSL port for the SOA admin server, and enter the name of the domain for WLS Domain.
6. Click Next.

7. In the Test dialog, click **Test Connection**.
8. Verify that the message `Success!` appears.
The application server connection has been created.

A

- abs function
 - description, B-8
- access points, 40-6
- access policies
 - on task content, 25-70
- action types, 25-45
- actionable emails, 29-30
- activities
 - annotations tab, A-2
 - assign, A-3
 - bind entity, A-4
 - compensate, A-5
 - correlation sets tab, A-2
 - create entity, A-6
 - definition, 5-4
 - email, A-6
 - empty, A-7
 - flow, A-8
 - flowN, A-9
 - IM, A-10
 - invoke, A-11
 - Java embedding, A-12
 - overview, 5-4, A-2
 - phase, A-12
 - pick, A-13
 - properties tab, A-2
 - receive, A-14
 - receive entity, A-16
 - receive signal, A-15
 - reply, A-16
 - scope, A-17
 - sensors tab, A-2
 - sequence, A-18
 - signal, A-19
 - SMS, A-19
 - switch, A-20
 - tasks common to many activities, A-2
 - terminate, A-21
 - throw, A-21
 - transform, A-22
 - user notification, A-23
 - voice, A-24
 - wait, A-24
 - while, A-25
- activity sensors
 - definition, 17-2
- Adapter Configuration wizard
 - starting, 5-10
- adapters
 - configuring, 5-10
 - definition, 5-10
 - in Oracle JDeveloper, 5-10
 - Oracle BAM, 32-1
 - overview, A-26
 - service names, 5-11
 - supported, 1-7
- add-dayTimeDuration-to-dateTime function
 - description, B-3
- adding a cross reference table column, 47-6
- adding columns to domain value maps, 46-7
- adding rows to domain value maps, 46-8
- addQuotes function
 - description, B-14
- ADF bindings filter, 30-2
- ADF task flow for human tasks, 26-3
- adfBindings bindings filter, 30-2
- adf-desktop-integration.jar, 30-2
- adfdiExcelDownload download filter, 30-3
- adfdiRemote servlet, 30-3
- ADFLibraryFilter filter, 30-3
- advanced formatting, message sources, 33-8
- aggregate functions in calculations, 31-4
- alerts
 - history, 37-6
 - Oracle BAM
 - about, 37-1
 - actions, F-5
 - activating, 37-3
 - activity, 37-6
 - conditions, F-5
 - creating, 37-2
 - dependencies, 37-6
 - events, F-1
 - frequency constraint, F-12
 - history, 37-6
 - messages, 37-5
 - parameterized, F-7
 - templates, 37-4
 - web services, 37-7
- annotations tab

- in activities, A-2
- ant scripts
 - managing composites, 43-22
- appendToList function
 - description, B-14
- application roles
 - definition, 24-5
- application template, 30-2
- AQ adapter
 - capabilities, A-27
- arrays
 - determining the size of, 7-29
 - in transformations, 45-27
 - manipulating, 7-28
 - maxOccurs attribute, 7-28
 - SOAP-encoded arrays not supported, 7-32
 - statically indexing into, 7-28
- assertion tests
 - overview, 49-2
- assertions
 - creating value asserts, 49-17
 - in composite test suites, 49-4
- assign activity
 - adding to an asynchronous service, 9-5
 - capabilities, A-3
 - copying data, 7-14
 - description, 7-2
 - for data manipulation, 7-2
 - formatting the email message body as
 - HTML, 16-8
 - in asynchronous services, 9-5
 - using multiple bpelx:append settings, A-4
- assignment service
 - configuration, 29-34
 - deploying a custom assignment service, 29-41
 - dynamic assignment functions, 29-34, 29-35, 29-36, 29-37
 - dynamically assigning task participants, 29-37
 - example of implementation, 29-39
 - implementing, 29-38
- asynchronous interaction with a notification timer
 - BPEL process as the client, 6-6
 - BPEL process as the service, 6-6
 - definition, 6-5
- asynchronous interaction with a timeout
 - BPEL process as the client, 6-5
 - BPEL process as the service, 6-5
 - definition, 6-4
- asynchronous interactions
 - BPEL process as the client, 6-4
 - BPEL process as the service, 6-4
 - definition, 6-3
 - returning faults, 12-24
- asynchronous processes
 - dehydration store, 9-9
- asynchronous services
 - assign activities, 9-5
 - calling, 9-2
 - correlation IDs, 9-8
 - invoke activities, 9-3, 9-7

- parallel flows, 10-1
- partner links, 9-2, 9-6, 9-7
- receive activities, 9-4, 9-7
- WS-Addressing, 9-8
- attachments
 - sending with the notification wizard, 16-7
 - task attachments with email notifications, 29-31
 - using MIME/DIME SOAP attachments, 7-36
 - using style sheets, 25-63
 - using WordML style sheets, 25-63
- attribute labels
 - internationalization, 29-16
- attributes
 - manipulating, 7-19
- auto mapping
 - in transformations, 45-30
 - with confirmation in transformations, 45-32

B

- batching
 - message batching limitations with Oracle Business Activity Monitoring, 32-8
- batchProcessActive function
 - description, B-33
- batchProcessCompleted function
 - description, B-33
- bind entity activity
 - capabilities, A-4
- binding components
 - definition, 1-6
 - supported, 4-10
- bindingFault
 - definition, 12-4
- boolean values
 - assigning, 7-18
- BPEL design environment
 - overview, 5-1
- BPEL files
 - definition, 2-6
- BPEL processes
 - common interaction patterns, 6-1, 22-1
 - creating, 4-6
 - definition, 1-2
- BPEL projects
 - naming conventions, 5-2
- BPEL sensor
 - Oracle BAM, 32-6
- BPEL XPath functions
 - examples, 7-4
- bpelx
 - exec extension
 - embedding SDOs, 13-7
 - in assign activities, A-4
- bpelx extensions
 - XML data manipulation, 7-20
- bpelx:append extension
 - appending data to a node list, B-14
 - description, 7-20
- bpelx:copyList extension

- copying a node list or a node, B-15
 - description, 7-26
- bpelx:exec extension
 - built-in methods, 13-3
- bpelx:headerVariable extension
 - description, 7-34
- bpelx:insertAfter extension
 - description, 7-22
- bpelx:insertBefore extension
 - description, 7-21
- bpelx:remove extension
 - description, 7-23
- bpelx:rename extension
 - description, 7-24
- bpelx:validate extension
 - description, 7-27
- building expression with domain value map
 - functions, 46-11
- business events
 - creating, 44-3
 - definition, 44-1
 - local and remote boundaries, 44-3
 - publishing, 44-9
 - specifying callback classes, 25-66
 - subscribing to, 44-6
- business faults
 - definition, 12-3
- business rules
 - action types, 25-45
 - fact types, 25-45
 - routing policies, 25-40
 - specifying advanced routing rules, 25-43
 - specifying advanced routing rules with business rules, 25-43
 - use case for data validation and constraint checks, 23-2
 - use case for dynamic processing, 23-2
 - use case for externalizing decision points in the process, 23-2
 - use case for human workflow, 23-2
 - use cases, 23-2
- Business Rules design environment
 - overview, 23-2
- Business Rules Designer
 - introduction, 2-18

C

- calculated fields, 31-4
- calculations
 - aggregate functions, 31-4
 - datetime functions, 31-4
 - expressions, 31-4
 - string functions, 31-4
- callback classes
 - specifying business events, 25-66
 - specifying on task status, 25-65
- callbacks
 - class loading, 29-41
 - task routing and customization in BPEL

- callbacks, 25-76
 - viewing, 25-87
- case sensitivity
 - human workflow, 29-45
- catch branch
 - creating, A-17
 - fault handling, 12-24
- channels
 - email, 16-5
 - IM, 16-8
 - SMS, 16-9
 - voice mail, 16-11
- class paths
 - for clients using local Enterprise JavaBeans, 28-7
 - for clients using remote Enterprise JavaBeans, 28-6
 - for clients using SOAP, 28-6
- clearing data objects, 31-17
- clearTaskAssignees function
 - description, B-45
- compare function
 - description, B-8
- compare-ignore-case function
 - description, B-9
- compensate activity
 - capabilities, A-5
 - definition, 12-31
 - fault handling, 12-31
- complex type
 - variables, 7-15
- Component Palette
 - introduction, 2-3, 2-9
- componentType file
 - definition, 2-2
- composite sensors
 - adding, 51-2
 - adding a property, 51-6
 - adding a variable, 51-5
 - adding an expression, 51-5
 - definition, 51-1
 - monitoring during runtime, 51-6
 - restrictions on use, 51-1
- composite test
 - assertions overview, 49-2
 - creating test suites, 49-5
 - creating value asserts, 49-17
 - definition, 49-1
 - deploying test suites, 49-23
 - emulating inbound messages, 49-8
 - emulations overview, 49-2
 - naming limitations on test suites and test cases, 49-5
 - test case overview, 49-1
 - test suite assertions, 49-4
 - test suite components, 49-2
 - test suite emulations, 49-3
 - test suites overview, 49-1
 - test suites process initiation, 49-3
 - XML assert, 49-2
- composite.xml file

- definition, 2-2, 4-4
- deployment descriptors, C-1, C-2
- registering sensors and sensor actions, 17-11
- syntax, 4-21
- concat function
 - description, 7-17
- conditional branching logic
 - definition, 11-1
 - use of XPath expressions, 11-1
 - using switch activities, 11-2
 - using while activities, 11-4
- conditional processing
 - with xsl choose, 45-26
 - with xsl if, 45-25
- configuration plans
 - creating, 43-33
 - creating with the WLST utility, 43-36
 - definition, 43-30
 - use cases, 43-32
- configuration properties
 - deployment descriptors, C-1
- connections
 - Oracle BAM Server, 32-2
- constant values
 - in transformations, 45-16
- copying security filters, 31-13
- copyList function
 - description, B-15
- core XPath functions
 - examples, 7-3
- correlation ID
 - WS-Addressing, 9-8
- correlation sets
 - associating with receive activities, 9-20
 - creating, 9-20
 - creating property aliases, 9-21
- correlation sets tab
 - in activities, A-2
- correlations
 - adding on an OnMessage branch of a pick activity, A-14
- countNodes function, 7-29
 - description, B-16
- create domain value maps, 46-4
- create entity activity
 - capabilities, A-6
- create instance
 - definition, 9-7
 - in receive activities, 9-7
- create-delimited-string function
 - description, B-9
- createInstance attribute, 9-8
- create-nodeset-from-delimited-string function
 - description, B-40
- createWordMLDocument function
 - description, B-45
- creating cross reference tables, 47-4
- creating folders for data objects, 31-9
- creating mediator component
 - mediator files, 18-4

- creating mediator service component
 - mediator files, 2-13
- cross reference table look up, 47-12
 - xref
 - lookupXRef function, 47-12
- cross reference tables, 47-1
 - adding a column, 47-6
 - creating, 47-3
 - deleting values, 47-16
 - looking up, 47-12
 - modifying, 47-3
 - populating columns, 47-6
 - xref
 - lookupXRef function, 47-12
 - markForDelete function, 47-16
 - populateXRefRow1M function, 47-9
- cross references
 - creating, 47-3
 - introduction, 47-1
 - modifying, 47-3
 - overview, 47-1
- current-date function
 - description, B-3
- current-dateTime function
 - description, B-4
- current-time function
 - description, B-4
- custom classes
 - adding to a SOA composite application, 13-5
- custom escalation function
 - using, 29-41
- custom sensors
 - publish type, 17-2

D

- data manipulation
 - accessing fields with complex type variables, 7-15
 - assigning boolean values, 7-18
 - assigning date or time, 7-18
 - assigning literal strings, 7-17
 - assigning numeric values, 7-16
 - concatenating strings, 7-17
 - converting from a string to a structured XML
 - object type, 7-33
 - copying data between variables, 7-14
 - determining array sizes, 7-29
 - dynamically indexing into a data sequence, 7-29
 - generating array-equivalent functionality with the
 - genEmptyElem function, 7-31
 - initializing variables, 7-13
 - manipulating arrays, 7-28
 - manipulating attributes, 7-19
 - mathematical calculations with XPath
 - functions, 7-16
 - statically indexing into a data sequence, 7-28
 - with assign activities, 7-2, 7-14
 - with XQuery and XSLT, 7-4
- data objects
 - about, 31-1

- adding dimensions, 31-14
- calculated column, 31-4
- clearing contents, 31-17
- contents, 31-9
- creating folders, 31-9
- datetime column, 31-5
- defining, 31-2
- deleting, 31-17
- dimensions, 31-14
- general information, 31-8
- indexes, 31-16
- layout, 31-8
- lookup column, 31-3
- moving, 31-16
- Oracle Data Integrator, 31-6
- organizing, 31-9
- permissions, 31-6
 - folders, 31-10
 - renaming, 31-16
 - security filters, 31-12
 - system, 31-5
 - viewing, 31-7
- data sequences
 - dynamically indexing into, 7-29
- database
 - sensor publish type, 17-2
- database adapter
 - capabilities, A-27
- database views
 - human workflow, 29-54
- DataObjectDefinition web service, 36-3
- DataObjectOperations web service, 36-2
- date time stamp field, 31-5
- dates
 - assigning, 7-18
- datetime functions in calculations, 31-4
- day-from-dateTime function
 - description, B-4
- defining a fault handler, 12-21
- dehydration store, 9-9
 - definition, 9-9
- deleting cross reference table value, 47-16
 - xref
 - markForDelete function, 47-16
- deleting data objects, 31-17
- deleting folders, 31-12
- demos
 - sensor actions, 17-3
 - sensors, 17-3
- deployment
 - creating an application server connection, 43-1
 - managing deployed composites, 4-24
 - of a single composite, 43-2
 - of an existing archive, 43-21
 - of multiple composites, 43-8
 - of shared metadata across composites, 43-10
 - of SOA composite applications, 4-26
 - with the ant scripts, 43-22
 - with the WLST utility, 43-22
 - deployment descriptor file
 - See web.xml file
- deployment descriptors
 - composite.xml file, C-1, C-2
 - configuration properties, C-1
 - defining a configuration property, C-1
 - deprecated, C-3
- dictionaries
 - in transformations, 45-35
- digital signatures, 29-17
 - acting on tasks that require a signature, 27-34
 - specifying, 25-68
- dimensions
 - adding to data objects, 31-14
 - data object, 31-14
 - time, 31-15
- doc function
 - description, B-16
- domain value maps
 - add columns, 46-7
 - add rows, 46-8
 - creation, 46-4
 - dvm
 - lookupValue function, 46-8
 - lookupValue1M function, 46-9
 - editing, 46-7
 - features, 46-2
 - one-to-many mapping, 46-4
 - qualifier order, 46-3
 - qualifiers, 46-2
 - one-to-many mapping, 46-4
 - qualifier order, 46-3
 - qualifiers, 46-2
 - using, 46-8
 - using in a transformation, 46-9
 - using lookupValue functions, 46-11
- domain value maps functions
 - dvm
 - lookupValue, 46-8
 - lookupValue1M, 46-9
- domain value maps qualifiers, 46-2
- download filter, 30-3
- dvm
 - lookupValue function, 46-8
 - lookupValue1M function, 46-9
- dynamic assignment functions
 - configuring, 29-36
 - configuring display names, 29-37
 - definition, 29-34
 - implementing, 29-35

E

- edit domain value maps, 46-7
 - add columns, 46-7
 - add rows, 46-8
- elements
 - ignoring in XSLT documents, 45-39
- email
 - dynamically setting addresses, 16-11
 - making emails actionable, 29-30

- notifications support, 16-2, 16-5
- email activity
 - capabilities, A-6
- email attachments
 - notifications support, 16-7
- email messages
 - HTML content for message body, 16-8
- empty activity
 - capabilities, A-7
 - definition, 12-30
 - fault handling, 12-30
- emulation tests
 - overview, 49-2
- emulations
 - emulating inbound messages, 49-8
 - in BPEL test suites, 49-3
- ending
 - tasks, 25-52
- ends-with function
 - description, B-9
- Enterprise JavaBeans
 - creating an Enterprise JavaBeans adapter service, 52-6
 - interacting with SOA composite applications, 52-1
 - support in workflow services, 29-1
- enterprise message sources
 - about, 33-1
 - creating, 33-2
 - datetime specification, 33-6
 - defining, 33-2, 35-1
 - XML formatting, 33-8
- entity variable
 - binding key, 7-9
 - creating, 7-6
 - definition, 7-5
 - samples, 7-6
 - using, 7-4
- error assignee
 - configuring, 25-50
 - definition, 24-7
- errors
 - invalid settings, A-32
- escalating
 - tasks, 25-52
- escalation policy
 - escalate after, 25-55
 - overview, 25-52, 25-53
 - specifying, 25-62
- evaluation time
 - definition, 17-4
- evidence store service, 29-17
 - definition, 29-17
 - Enterprise JavaBeans, SOAP, and Java support, 29-2
 - WSDL file location, 29-3
- Excel workbook
 - MIME mapping, 30-3
- exceptions, 12-3
- expiration policy

- expire after, 25-54
- never expire, 25-53
- overview, 25-52, 25-53
- renew after, 25-54
- export file sample
 - ICommand, G-18
- expression builder dialog
 - using domain value map functions, 46-11
- expression constants
 - variable initialization, 7-13
- expressions in calculations, 31-4
- external data source
 - about, 35-1
 - creating, 35-1
 - Oracle Data Integrator, 35-2
- external routing
 - routing policy, 25-49

F

- fact types, 25-45
- fault bindings, 20-6
- fault handling, 12-21
 - creating, 12-1, 12-21
 - definition, 12-1
 - importing RuntimeFault.wsdl, 12-21
 - modifying the WSDL files, 12-21
 - returning external faults, 12-23, 12-24
 - throwing internal faults, 12-22
 - using catch branches, 12-24
 - using compensate activities, 12-31
 - using empty activities, 12-30
 - using scope activities, 12-24
 - using terminate activities, 12-33
 - using the getFaultAsString function, 12-21
 - using throw activities, 12-22
- fault management framework
 - associating a fault policy with a fault policy binding, 12-10
 - definition, 12-4
 - designing, 12-5
 - executing a fault policy, 12-15
 - using a Java action fault policy, 12-15
- fault policy, 20-1
 - actions, 20-4
 - associating with a fault policy binding, 12-10
 - component level, 20-6
 - composite level, 20-6
 - conditions, 20-2
 - definition, 12-4
 - designing, 12-5
 - executing, 12-15
 - sample file, 12-9
 - using a Java action fault policy, 12-15
- fault policy bindings
 - sample file, 12-14
- fault sensors
 - definition, 17-2
- fault-bindings.xml, 20-12
 - fault policy bindings file, 12-5

- fault-policies.xml, 20-8
 - fault policy file, 12-5
- faults
 - categories of faults in BPEL, 12-3
 - Qname fault name, 12-3
 - returning external faults, 12-23, 12-24
 - standard faults, 12-3
 - throwing internal faults, 12-22
- fields
 - calculated, 31-4
 - lookup, 31-3
 - timestamp, 31-5
- file adapter
 - capabilities, A-27
- Filter
 - adfBindings, 30-2
 - adfdiExcelDownload, 30-3
 - ADFLibraryFilter, 30-3
 - bindings filter, 30-2
- filters
 - copying, 31-13
 - Oracle BAM security, 31-12
- fire and forget
 - one-way message, 6-1
- flex fields
 - using, 27-52
 - values, 29-15
- flow activity
 - capabilities, A-8
- flowN activity
 - capabilities, A-9
 - definition, 10-5
- folder permissions, 31-10
- folders
 - deleting, 31-12
 - renaming, 31-11
- format function
 - description, B-33
- formatDate function
 - description, B-18
- format-dateTime function
 - description, B-5
- format-string function
 - description, B-10
- FTP adapter
 - capabilities, A-27
- functions
 - abs, B-8
 - add-dayTimeDuration-to-dateTime, B-3
 - addQuotes, B-14
 - appendToList, B-14
 - batchProcessActive, B-33
 - batchProcessCompleted, B-33
 - chaining in transformations, 45-19
 - clearTaskAssignees, B-45
 - compare, B-8
 - compare-ignore-case, B-9
 - concat, 7-17
 - copyList, B-15
 - countNodes, 7-29, B-16
 - create-delimited-string, B-9
 - create-nodeset-from-delimited-string, B-40
 - createWordMLDocument, B-45
 - creating user-defined XPath extension functions, B-59
 - current-date, B-3
 - current-dateTime, B-4
 - current-time, B-4
 - day-from-dateTime, B-4
 - descriptions, 45-17
 - doc, B-16
 - dynamically setting email addresses and telephone numbers, 16-11
 - editing in transformations, 45-18
 - editing XPath expressions in transformations, 45-22
 - ends-with, B-9
 - examples, 7-3
 - format, B-33
 - formatDate, B-18
 - format-dateTime, B-5
 - format-string, B-10
 - functions prefixed with xp20 or orcl, 45-17
 - genEmptyElem, 7-31, B-34
 - generateGUID, B-19
 - generate-guid, B-40
 - getChildElement, B-34
 - getContentAsString, B-21
 - get-content-as-string, B-10
 - getConversationId, B-21
 - getCreator, B-21
 - getCurrentDate, 7-18, B-22
 - getCurrentDateTime, 7-18, B-22
 - getCurrentTime, 7-18, B-22
 - getDefaultRealmName, B-49
 - getDomainId, B-22
 - getElement, B-23
 - getFaultAsString, 12-21
 - getGroupIdsFromGroupAlias, B-24
 - getGroupProperty, B-50
 - getInstanceId, B-24
 - getLinkStatus, B-31
 - get-localized-string, B-11
 - getManager, B-50
 - getMessage, B-34
 - getNodes, B-25
 - getNodeValue, B-24
 - getNotificationProperty, B-45
 - getNumberOfTaskApprovals, B-46
 - getPreference, B-25
 - getPreviousTaskApprover, B-46
 - getProcessId, B-26
 - getProcessOwnerId, B-26
 - getProcessURL, B-26
 - getProcessVersion, B-26
 - getReportees, B-50
 - getTaskAttachmentByIndex, B-46
 - getTaskAttachmentByName, B-47
 - getTaskAttachmentContents, B-47
 - getTaskAttachmentsCount, B-47

- getTaskResourceBindingString, B-47
- getUserAliasId, B-27
- getUserProperty, 16-12, B-51
- getUserRoles, B-52
- getUsersInGroup, B-52
- getVariableData, 16-12, B-32
- getVariableProperty, B-32
- hours-from-dateTime, B-5
- implicit-timezone, B-5
- in transformations, 45-17
- index-within-string, B-11
- integer, B-28
- isUserInRole, B-53
- last-index-within-string, B-12
- left-trim, B-12
- location of function descriptions, 7-4
- lookupGroup, B-53
- lookup-table, B-1
- lookupUser, B-53
- lookup-xml, B-43
- lower-case, B-13
- matches, B-13
- max-value-among-nodeset, B-35
- minutes-from-dateTime, B-6
- min-value-among-nodeset, B-35
- month-from-dateTime, B-6
- parseEscapedXML, 7-33, B-28
- position, 7-29
- prefixed with xp20 or orcl, 45-17
- processXQuery, B-29
- processXSLT, 16-8, B-29
- processXSQL, B-30
- query-database, B-2
- readBinaryFromFile, B-30
- readFile, B-30
- right-trim, B-13
- seconds-from-dateTime, B-6
- selecting an data sequence element, 7-28
- sequence-next-val, B-2
- square-root, B-36
- subtract-dayTimeDuration-from-dateTime, B-6
- timezone-from-dateTime, B-7
- translateFromNative, B-36
- translateToNative, B-36
- upper-case, B-14
- wfDynamicGroupAssign, B-48
- wfDynamicUserAssign, B-49
- writeBinaryToFile, B-31
- year-from-dateTime, B-7
- Fusion Web Application (ADF) application
 - template, 30-2
- FYI assignee
 - configuring, 25-37
 - definition, 24-5, 25-37
 - workflow participant type, 24-5, 25-37

G

- genEmptyElem function
 - description, 7-31, B-34

- generateGUID function
 - description, B-19
- generate-guid function
 - description, B-40
- getChildElement function
 - description, B-34
- getContentAsString function
 - description, B-21
- get-content-as-string function
 - description, B-10
- getConversationId function
 - description, B-21
- getCreator function
 - description, B-21
- getCurrentDate function
 - description, 7-18, B-22
- getCurrentDateTime function
 - description, 7-18, B-22
- getCurrentTime function
 - description, 7-18, B-22
- getDefaultRealmName function
 - description, B-49
- getDomainId function
 - description, B-22
- getElement function
 - description, B-23
- getFaultAsString function
 - description, 12-21
- getGroupIdsFromGroupAlias function
 - description, B-24
- getGroupProperty function
 - description, B-50
- getInstanceId function
 - description, B-24
- getLinkStatus function
 - description, B-31
- get-localized-string function
 - description, B-11
- getManager function
 - description, B-50
- getMessage function
 - description, B-34
- getNodes function
 - description, B-25
- getNodeValue function
 - description, B-24
- getNotificationProperty function
 - description, B-45
- getNumberOfTaskApprovals function
 - description, B-46
- getPreference function
 - description, B-25
- getPreviousTaskApprover function
 - description, B-46
- getProcessId function
 - description, B-26
- getProcessOwnerId function
 - description, B-26
- getProcessURL function
 - description, B-26

- getProcessVersion function
 - description, B-26
- getReportees function
 - description, B-50
- getTaskAttachmentByIndex function
 - description, B-46
- getTaskAttachmentByName function
 - description, B-47
- getTaskAttachmentContents function
 - description, B-47
- getTaskAttachmentsCount function
 - description, B-47
- getTaskResourceBindingString function
 - description, B-47
- getUserAliasId function
 - description, B-27
- getUserProperty function
 - description, B-51
 - example, 16-12
- getUserRoles function
 - description, B-52
- getUsersInGroup function
 - description, B-52
- getVariableData function
 - description, 7-17, B-32
 - example, 16-12
 - using in mathematical calculations, 7-16
- getVariableProperty function
 - description, B-32
- global task variable name
 - specifying in human task activities, 25-83
- group vote
 - configuring, 25-30
 - consensus percentage, 25-32
 - immediately triggering a voted outcome when a minimum percentage is met, 25-32
 - specifying group voting details, 25-32
 - waiting until all votes are in before triggering an outcome, 25-33

H

- headers
 - SOAP headers, 7-34
- heap size
 - increasing, 45-47
- hours-from-dateTime function
 - description, B-5
- human task activity
 - associating with a BPEL process, 25-77
 - identification key, 25-83
 - including the task history of other tasks, 25-84
 - scope name and global task variable name, 25-83
 - specifying a task initiator and task priority, 25-80
 - specifying a task title, 25-79
 - specifying task parameters, 25-80
 - task owner, 25-83
 - viewing BPEL callbacks, 25-87
- human task definition
 - associating with a BPEL process, 25-2

- Human Task Editor
 - abruptly completing a condition, 25-41
 - accessing the sections of, 25-6
 - actionable emails, 29-30
 - allowing all participants to invite other participants, 25-41
 - assigning task participants by name or expression, 25-25, 25-52
 - bypassing task participants, 25-30, 25-34, 25-37
 - configuring the error assignee, 25-50
 - creating a human task, 25-3
 - editing notification messages, 25-59
 - escalate after policy, 25-55
 - escalating, renewing, or ending a task, 25-52
 - escalation and expiration policy overview, 25-52, 25-53
 - escalation rules, 25-62
 - expire after policy, 25-54
 - FYI assignee task participant, 25-37
 - group voting details, 25-32
 - introduction, 2-14
 - inviting additional task participants, 25-30, 25-34, 25-36
 - multilingual settings, 25-63, 29-29
 - never expire policy, 25-53
 - notification preferences, 25-56
 - notifying recipients of changes to task status, 25-57
 - parallel task participant, 25-30
 - renew after policy, 25-54
 - securing notifications, 29-32
 - serial task participant, 25-34
 - setting up reminders, 25-60
 - sharing attachments and comments with task participants, 25-33
 - single approver task participant, 25-22
 - specifying access policies, 25-70
 - specifying business event callbacks, 25-66
 - specifying callback classes, 25-65
 - specifying digital signatures, 25-68
 - style sheets in attachments, 25-63
 - task attachments with email notifications, 29-31
 - task category, 25-10
 - task outcome, 25-8
 - task owner specification through the user directory, 25-11
 - task owner specification through XPath expressions, 25-15
 - task participants, 25-18
 - task payload data structure, 25-16
 - task priority, 25-10
 - task routing and customization in BPEL callbacks, 25-76
 - task title, 25-8
 - time limits for acting on tasks, 25-29, 25-33, 25-36
 - WordML style sheets in attachments, 25-63
- human tasks
 - creating, 25-3
 - designing a human task, 24-13
- human workflow

- access rules, 24-9
- application roles, 24-5
- case sensitivity, 29-45
- concepts, 24-3
- database views, 29-54
- definition, 24-1
- groups, 24-5
- integration with Oracle WebLogic Server, 29-45
- participant assignments, 24-5
- participant types, 24-4
- participants, 24-4
- routing policies, 25-38
- task assignments, 24-6
- task deadlines, 24-8
- task stakeholders, 24-7
- use cases, 24-11
- users, 24-5

I

ICommand

- clear, G-3
- command line, 38-5
- delete, G-3
- detailed command descriptions, G-3
- export, G-4
 - sample, G-18
- general command and option syntax, 38-2
- import, G-10
- log, G-17
- operations, G-1
- regular expressions, G-18
- remote execution, 38-6
- rename, G-14
- running, 38-1
- sample export file, G-18
- summary of commands, G-1
- syntax, 38-2
- syntax, object names, 38-3
- XML file, G-15

ICommand utility, 38-1

ICommand web service, 36-4

identification key

- specifying in human task activities, 25-83

identity service

- definition, 24-28, 29-9
- determining a user's local language and time zone, 27-62

- Enterprise JavaBeans, SOAP, and Java support, 29-2

functions

- getDefaultRealmName, B-49
- getGroupProperty, B-50
- getManager, B-50
- getReportees, B-50
- getUserProperty, B-51
- getUserRoles, B-52
- getUsersInGroup, B-52
- isUserInRole, B-53
- lookupGroup, B-53

- lookupUser, B-53
- providers, 29-10, 29-11
- support for in workflows, 29-9
- supported task operations, 29-9
- use with JAZN, 29-9, 29-10
- use with LDAP, 29-9, 29-10
- WSDL file location, 29-2

IM activity

- capabilities, A-10
- notifications support, 16-8

implicit-timezone function

- description, B-5

import

- source and target schemas into a transformation, 45-8
- two schema files of the same name into the same project is not supported, 2-11

indexes

- in data objects, 31-16

indexing methods

- using XPath, 7-29

index-within-string function

- description, B-11

instances

- starting new, 9-8

integer function

- description, B-28

interaction patterns

- asynchronous interaction with a notification timer, 6-5
- asynchronous interaction with a timeout, 6-4
- asynchronous interactions, 6-3
- common patterns between a BPEL process and another application, 6-1, 22-1
- multiple interactions, 6-10
- one request, a mandatory response, and an optional response, 6-8
- one request, multiple responses, 6-6
- one request, one of two possible responses, 6-7
- one-way message, 6-1
- partial processing, 6-9
- synchronous interactions, 6-2

Invalid Settings error message, A-32

invoke activity

- adding to an asynchronous service, 9-3
- capabilities, A-11
- definition, 5-5, 8-1
- in asynchronous services, 9-3, 9-7
- in synchronous services, 8-1, 8-5

isUserInRole function

- description, B-53

J

JAR

- See* .JAR Files

.JAR files

- adding custom classes and JAR files, 13-5
- adf-desktop-integration.jar, 30-2
- resourcebundle.jar file, 30-2

- wscclient.jar, 30-2
- Java
 - support in workflow services, 29-1
- Java applications
 - wrapped as SOAP services, 13-1
- Java Connector Architecture (JCA)
 - definition, 1-2
- Java embedding
 - bpel:exec extension, 13-3
 - example, 13-6
 - in a BPEL process, 13-1
- Java embedding activity
 - capabilities, A-12
- JAZN
 - storing a user's local language and time zone, 27-62
 - use with identity service, 29-9, 29-10
- JMS
 - definition, 1-2
- JMS adapter
 - capabilities, A-27
 - sensor publish type, 17-2
- JMS queue
 - sensor publish type, 17-2
- JMS topic
 - sensor publish type, 17-2

K

- knowledge module
 - Oracle BAM, 34-2

L

- languages
 - changing, 27-62
 - preferences, 27-62
 - setting in JAZN, 27-62
 - setting in LDAP, 27-62
- large documents
 - best practices for handling, 53-1
 - importing large data sets in Oracle B2B, 53-6
 - large numbers of mediators in composites, 53-6
 - limitations on concurrent processing, 53-6
 - opaque schema for processing large payloads, 53-6
 - processing, 53-1
 - processing in Oracle B2B, 53-3
 - setting a default JTA timeout for large documents, 53-5
 - setting audit levels, 53-2
 - streaming MTOM attachments, 53-6
 - using a flow with multiple sequences, 53-5
 - using a flow with no sequence, 53-6
 - using a flow with one sequence, 53-5
 - using adapter support for streaming large payloads, 53-2
 - using assign activities in BPEL and mediator, 53-2
 - using correct settings for large payload

- scenarios, 53-3
- using large numbers of activities in BPEL processes (with FlowN), 53-5
- using large numbers of activities in BPEL processes (without FlowN), 53-5
- using XSLT transformations for repeating structures, 53-2
- using XSLT transformations on large payloads (for BPEL and mediator), 53-6
- last-index-within-string function
 - description, B-12
- layouts, data object, 31-8
- LDAP
 - storing a user's local language and time zone, 27-62
 - used with identity service, 29-9, 29-10
- left-trim function
 - description, B-12
- literal strings
 - assigning, 7-17
- literal XML
 - variable initialization, 7-13
- localization, worklist, 27-62
- looking up cross reference tables, 47-12
 - xref
 - lookupXRef function, 47-12
- lookup fields, 31-3
- lookupGroup function
 - description, B-53
- lookup-table function
 - description, B-1
- lookupUser function
 - description, B-53
- lookupValue functions
 - dvm
 - lookupValue function, 46-8
 - lookupValueIM function, 46-9
- lookup-xml function
 - description, B-43
- lower-case function
 - description, B-13

M

- management chains
 - participant lists, 25-26
- ManualRuleFire web service, 36-4
- map parameters
 - creating in transformations, 45-36
- map variables
 - creating in transformations, 45-36
- master and detail processes
 - creating, 15-6
 - definition, 15-1
 - receive signal activity, A-15
 - signal activity, A-19
- matches function
 - description, B-13
- maxOccurs attribute, 7-28, 7-29
 - setting for transformations, 45-48

- max-value-among-nodeset function
 - description, B-35
- mediator creation
 - specifying operation or event subscription properties, 18-25
- mediator files
 - .componentType, 2-13, 18-5
 - composite.xml, 2-13, 18-5
 - .mplan, 2-13, 18-5
 - .wsdl, 2-13, 18-5
- mediator service component
 - mediator files, 2-13, 18-4
- message filtering, 40-7
- message schemas
 - updating, 4-21, 4-22
- message source advanced formatting, 33-8
- message sources, 33-1
- MessageFilter, 40-7
- MessageFilterFactory, 40-7
- messages
 - receiving, 40-6
 - rejecting, 40-7
- MessagingClientFactory, 40-2
- MessagingClient.receive, 40-7
- MessagingClient.registerAccessPoint, 40-6
- MessagingClient.registerMessageFilter, 40-7
- metadata
 - service components, 23-10
- Metadata Service (MDS)
 - definition, 1-6
- MIME mapping
 - Excel workbook, 30-3
- minOccurs attribute
 - setting for transformations, 45-48
- minutes-from-dateTime function
 - description, B-6
- min-value-among-nodeset function
 - description, B-35
- modes
 - xref
 - populateXRefRow function, 47-7
 - populateXRefRow1M function, 47-9
- modifying a mediator, 18-25
 - modifying event subscriptions, 18-27
 - modifying operations, 18-25
- modifying cross reference tables
 - adding a column, 47-6
- modifying mediator event subscriptions, 18-27
- modifying mediator operations, 18-25
- month-from-dateTime function
 - description, B-6
- MQ adapter
 - capabilities, A-28
- multilingual settings
 - specifying in tasks, 25-63, 29-29
- myRole attribute
 - definition, 9-7

N

- named templates
 - creating, 45-19
 - in functions, 45-19
- names and expressions
 - participant list, 25-24
- naming conventions
 - for BPEL projects, 5-2
- normalized message properties
 - Oracle BPEL Process Manager, H-1
 - Oracle Web Services Addressing, H-2
- NOT operator, 31-4
- notification messages
 - editing, 25-59
- notification services
 - actionable emails, 29-30
 - configuring the notification channel, 29-28
 - custom notification headers, 29-34
 - definition, 24-28
 - error message support, 29-27
 - multilingual settings, 29-29
 - notification contents, 29-26
 - reliability support, 29-27
 - sending inbound and outbound
 - attachments, 29-31
 - sending inbound comments, 29-32
 - sending reminders, 29-32
 - sending secure notifications, 29-32
 - setting automatic replies to unprocessed messages, 29-33
 - specifying participant notification preferences, 25-56
- notifications
 - allowing the end user to select the notification channels, 16-13
 - configuring in Oracle JDeveloper, 16-3
 - definition, 24-8
 - dynamically setting email addresses and telephone numbers, 16-11
 - email attachment support, 16-7
 - email support, 16-2, 16-5
 - formatting the email message body as
 - HTML, 16-8
 - IM support, 16-8
 - selecting recipients by browsing the user directory, 16-12
 - setting up, 16-3
 - SMS support, 16-9
 - voice mail support, 16-11
- notifications and reminders
 - in tasks, 29-25
- numeric values
 - assigning, 7-16

O

- onAlarm branch
 - of pick activity, 14-2
- one-to-many mapping, 46-4
- onMessage branch

- of pick activity, 14-2
- operators
 - AND operator, 31-4
 - OR operator, 31-4
- Oracle Application Development Framework (ADF)
 - binding component, 1-7
- Oracle Applications adapter
 - capabilities, A-28
- Oracle B2B
 - capabilities, A-27
- Oracle BAM, 32-6
 - See* Oracle Business Activity Monitoring
- Oracle BAM Adapter, 32-1
- Oracle BAM knowledge modules, 34-2
- Oracle BAM Server
 - creating a BPEL sensor, 32-6
 - creating a BPEL sensor action, 32-7
 - creating a connection to, 32-2
- Oracle BAM Server connection, 32-2
- Oracle BPEL Designer
 - introduction, 2-4
- Oracle BPM Worklist
 - See* worklist
- Oracle Business Activity Monitoring
 - capabilities, A-27
 - creating a BPEL sensor action for Oracle BAM Server, 32-7
 - creating a BPEL sensor for Oracle BAM Server, 32-6
 - creating a connection to Oracle BAM Server, 32-2
 - definition
 - integration with Oracle BPEL Process Manager sensors, 32-6
 - message batching limitations, 32-8
 - overview, 32-6
- Oracle Enterprise Manager
 - introduction, 2-20
- Oracle Internet Directory
 - storing a user's local language and time zone, 27-62
- Oracle JDeveloper
 - adapters, 5-10
 - adding the SOA extensions, 4-2
 - Component Palette, 2-9
 - configuring notifications, 16-3
 - creating sensors, 17-3
 - overview of design environment, 5-1
 - overview of rules designer environment, 23-2
 - services, 2-9
 - transformations, 45-6
- Oracle JDeveloper project
 - desktop integration, adding, 30-2
- Oracle Mediator
 - define routing rules, 19-1
 - definition, 18-1
 - routing rules, 19-1
- Oracle Mediator component creation
 - mediator files, 2-13, 18-4
- Oracle Mediator Editor, 18-5
 - environment

- Application Navigator, 18-4
- History Window, 18-5
- Log Window, 18-6
- Oracle Mediator Editor, 18-5
- Property Inspector, 18-6
- Source View, 18-5
- Structure Window, 18-6
- introduction, 2-11
- Oracle Mediator error handling
 - actions, 20-4
 - conditions, 20-2
 - fault bindings, 20-6
 - fault policy, 20-1
 - introduction, 20-1
 - using, 20-7
 - XML schema files, 20-8
- Oracle Service Registry
 - publishing and browsing, A-29
- Oracle SOA Suite
 - introduction, 1-1
- Oracle User Messaging Service (UMS)
 - configuring, 39-1
 - definition, 16-2
- organizing data objects, 31-9
- overview, 17-2

P

- parallel
 - definition, 25-30
 - workflow participant type, 25-30
- parallel blocks
 - definition, 25-19
- parallel flows
 - definition, 10-1
- parseEscapedXML function
 - description, 7-33, B-28
- partial processing
 - BPEL process as the client, 6-10
 - BPEL process as the service, 6-10
 - definition, 6-9
- participant assignments
 - definition, 24-5
- participant lists
 - rulesets, 25-27
 - value-based management chains, 25-26
 - value-based names and expressions, 25-24
- participant types
 - FYI assignee, 24-5, 25-37
 - parallel, 24-4, 25-30
 - serial, 24-4, 25-34
 - single approver, 24-4, 25-22
- partner links
 - adding to an asynchronous service, 9-2
 - capabilities, A-28
 - creating, 5-6
 - definition, 5-5
 - in asynchronous services, 9-2, 9-6, 9-7
 - in synchronous services, 8-1
 - Oracle BAM, 32-4

- overview, 5-5
- specifying a WSDL file, 5-6
- partnerLinkType
 - definition, 9-6
- partnerRole attribute
 - definition, 9-7
- patterns
 - of interaction between a BPEL process and another application, 6-1, 22-1
- permissions
 - copying, 31-7
 - data objects, 31-6
 - setting on folders, 31-10
- phase activity
 - capabilities, A-12
- pick activity
 - adding correlations on an OnMessage
 - branch, A-14
 - capabilities, A-13
 - code example, 14-4
 - condition branches, 14-1
 - for timeouts, 14-1
 - onAlarm branch, 14-2
 - onMessage branch, 14-2
- policies
 - attaching, 50-2
 - definition, 50-1
 - supported categories, 50-1
- populating cross reference tables, 47-6
 - xref
 - populateXRefRow1M function, 47-9
- ports
 - in synchronous services, 8-1
- portType
 - definition, 9-6
- position function
 - description, 7-29
- process initiation
 - in BPEL test suites, 49-3
- processes
 - naming conventions, 5-2
- processXQuery function
 - description, B-29
- processXSLT function
 - description, B-29
 - example, 16-8
- processXSQL function
 - description, B-30
- projects
 - BPEL file, 2-6
 - importing two schema files of the same name into the same project is not supported, 2-11
 - naming conventions, 5-2
 - ViewController, 30-2
 - WSDL file, 2-6
- properties tab
 - in activities, A-2
- property aliases
 - creating for correlation sets, 9-21
- public views

- sensors, D-1
- publish types
 - creating a custom publisher, 17-9
 - custom, 17-2
 - database, 17-2
 - definition, 17-2
 - JMS Adapter, 17-2
 - JMS queue, 17-2
 - JMS topic, 17-2

Q

- Qname
 - fault name, 12-3
- qualifier, 46-2
 - qualifier order, 46-3
- qualifier order, 46-3
- query-database function
 - description, B-2

R

- readBinaryFromFile function
 - description, B-30
- readFile function
 - description, B-30
 - reading files from absolute directory paths, B-30
- receive activity
 - adding to an asynchronous service, 9-4
 - associating with correlation sets, 9-20
 - capabilities, A-14
 - create instance, 9-7
 - creating new instances, 9-8
 - in asynchronous services, 9-4, 9-7
- receive entity activity
 - capabilities, A-16
- receive signal activity
 - capabilities, A-15
- receiving a message, 40-6
- references
 - adding, 4-17, 4-19
 - definition, 1-6, 4-11
 - deleting, 4-19
 - wiring, 4-19
- regular expressions
 - ICommand, G-18
- rejecting messages, 40-7
- reminders
 - for task notifications, 29-32
- remoteFault
 - definition, 12-4
- renaming data objects, 31-16
- renaming folders, 31-11
- renewing
 - tasks, 25-52
- repeating elements
 - in transformations, 45-27
- replayFault
 - definition, 12-4
- reply activity

- capabilities, A-16
- reporting schema
 - for database publish type of sensors, D-1
- reports
 - correcting memory errors when generating for transformations, 45-47
 - customizing sample XML generation for transformations, 45-48
 - generating for transformations, 45-46
 - worklist, 27-57
- resource bundles, 29-42
 - class loading, 29-41
 - for displaying tasks in different languages, 25-63, 29-29
- Resource Palette
 - introduction, 2-3
 - using, 4-12
- resourcebundle.jar file, 30-2
- revisions
 - invoking the default revision, 4-15
 - setting the default revision, 4-26
- right-trim function
 - description, B-13
- roles
 - for partner links in asynchronous services, 9-6
- routing policies
 - available types, 25-40
 - business rules, 25-40
 - completing parent subtasks of early completing subtasks, 25-43
 - enabling early completion in parallel subtasks, 25-43
 - external routing, 25-40, 25-49
 - routing a task to all participants in the order specified, 25-40
 - selecting, 25-38
- routing rules, 19-1
 - define, 19-1
 - defining, 19-1
 - filter expression, 19-15
 - introduction, 19-1
- routing slip
 - definition, 25-29
- RPC styles
 - differences with document-literal styles in WSDL files, 7-1, 7-33
- rules
 - service component, 23-10
- rulesets
 - participant lists, 25-27
- runtime config service
 - definition, 24-29
 - Enterprise JavaBeans, SOAP, and Java support, 29-2
 - supported task operations, 29-14
 - WSDL file location, 29-3
- runtime exceptions, 12-3
- runtime faults
 - definition, 12-3
 - example, 12-21

- RuntimeFault.wsdl file
 - importing into a process, 12-21

S

- SAR
 - See* SOA archive (SAR)
- SCA technologies
 - introduction, 1-4
- schema files
 - creating a transformation map file from imported schemas, 45-8
 - replacing in the XSLT Mapper, 45-39
- schemas
 - updating message schemas, 4-21, 4-22
- scope activity
 - capabilities, A-17
 - creating, 12-25
 - fault handling, 12-24
 - using a fault handler in a scope activity, 12-27
- scope name
 - specifying in human task activities, 25-83
- seconds-from-dateTime function
 - description, B-6
- security filters
 - copying, 31-13
 - on data objects, 31-12
- security model
 - for workflow services, 29-3
 - in SOAP web services, 29-4
 - workflow context on behalf of a user, 29-4
- sensor actions
 - configuring, 17-6
 - creating a BPEL sensor action for Oracle BAM Server monitoring, 32-7
 - demos, 17-3
 - viewing metadata, 17-12
 - XSD schema file, D-5
- sensor data
 - persisting in a reporting schema, D-1
- sensors, 17-2, 32-6
 - activity sensors, 17-2
 - BPEL reporting schema, D-1
 - configuring, 17-3
 - creating a BPEL sensor for Oracle BAM Server to monitor, 32-6
 - creating a connection to Oracle BAM Server, 32-2
 - creating a custom publish type, 17-9
 - creating in Oracle JDeveloper, 17-3
 - definition, 17-2
 - demos, 17-3
 - evaluation time, 17-4
 - fault sensors, 17-2
 - integration with Oracle Business Activity Monitoring, 32-6
 - public views, D-1
 - publish types, 17-2
 - sensor actions XSD schema file, D-5
 - variable sensors, 17-2
 - viewing metadata, 17-12

- sensors tab
 - in activities, A-2
- sequence activity
 - capabilities, A-18
- sequence-next-val function
 - description, B-2
- sequential blocks
 - definition, 25-19
- sequential list of approvers
 - configuring, 25-34
- serial
 - definition, 25-34
 - workflow participant type, 25-34
- Service Component Architecture (SCA)
 - definition, 1-2
- service components
 - adding, 4-5, 4-7
 - available types, 1-7
 - BPEL process, 1-8
 - business rules, 1-8
 - deleting, 4-7
 - editing, 4-8
 - human task, 1-8
 - introduction, 4-5
 - mediator, 1-8
 - metadata, 23-10
 - rules, 23-10
 - web service, 23-10
 - wiring, 4-15, 4-19
- Service Data Objects (SDO), 7-6
 - definition, 1-2
 - converting from XML to SDO, 7-12
 - declaring SDO-based variables, 7-11
 - embedding with bpelx
 - exec, 13-7
 - entity variable support, 7-6
 - passing parameters between Enterprise JavaBeans and SOA composite applications, 52-1
 - using in an Enterprise JavaBeans application, 52-2
 - using standalone SDO-based variables, 7-11
- service engines
 - definition, 1-6
 - human workflow, 24-31
- service infrastructure
 - definition, 1-6
- service names
 - in adapters, 5-11
- services
 - adding, 4-9, 4-14
 - AQ adapter, A-27
 - automatically exposing as a SOAP service, 4-10
 - database adapter, A-27
 - definition, 1-6, 2-9, 4-11
 - deleting, 4-14
 - file adapter, A-27
 - FTP adapter, A-27
 - JMS adapter, A-27
 - MQ adapter, A-28
 - Oracle Applications adapter, A-28
 - Oracle B2B, A-27
 - Oracle Business Activity Monitoring, A-27
 - overview, A-26
 - partner link, A-28
 - selecting a WSDL, 4-11
 - socket adapter, A-29
 - wiring, 4-15
- servlet
 - adfdiRemote, 30-3
- setting folder permissions, 31-10
- setting up, 29-30
- signal activity
 - capabilities, A-19
- single approver
 - configuring, 25-22
 - definition, 25-22
 - workflow participant type, 25-22
- SMS activity
 - capabilities, A-19
 - notifications support, 16-9
- SOA archive (SAR)
 - deploying, 43-3
- SOA composite applications
 - activating, 4-26
 - creating, 4-1
 - definition, 1-3
 - deploying a single composite, 43-2
 - deploying an existing archive, 43-21
 - deploying multiple composites, 43-8
 - deploying shared metadata across composites, 43-10
 - deployment, 4-26
 - interacting with Enterprise JavaBeans, 52-1
 - invoking other composites, 4-23
 - invoking the default revision, 4-15
 - retiring, 4-26
 - setting as the default revision, 4-26
 - shutting down, 4-25
 - starting up, 4-25
 - testing, 4-27
 - undeploying, 4-26
- SOA Composite Editor
 - introduction, 2-1, 4-1
 - overview, 4-1
- SOA extensions
 - adding to Oracle JDeveloper, 4-2
- SOA project
 - creating, 4-1
- SOAP
 - definition, 1-2
 - security in SOAP web services, 29-4
 - support in workflow services, 29-1
 - using MIME/DIME attachments, 7-36
- SOAP headers, 7-34
 - receiving in BPEL, 7-35
 - sending in BPEL, 7-36
- SOAP services
 - performance issues, 13-1
 - using Java code, 13-1
- SOAP-encoded arrays

- not supported, 7-32
- socket adapter
 - capabilities, A-29
- sources
 - message, 33-1
- specifying operation or event subscription
 - properties, 18-25
 - validate syntax (XSD), 18-25
- square-root function
 - description, B-36
- stages
 - definition, 25-19
- standard faults
 - definition, 12-3
- string functions in calculations, 31-4
- strings
 - concatenating, 7-17
 - converting to an XML element, 7-33
- style sheets
 - using for attachments, 25-63
- subtract-dayTimeDuration-from-dateTime function
 - description, B-6
- switch activity
 - capabilities, A-20
 - in conditional branching logic, 11-2
- synchronous callbacks, 8-1
 - operational concepts, 8-2
 - syncMaxWaitTime property, 8-2
- synchronous interactions
 - BPEL process as the client, 6-3
 - BPEL process as the service, 6-3
 - definition, 6-2
 - returning faults, 12-23
- synchronous processes
 - calling a one-way mediator, 8-6
- synchronous receiving, 40-7
- synchronous services
 - callbacks with the partner link and invoke activity, 8-1
 - calling, 8-2
 - invoke activities, 8-5
 - ports, 8-1
- syncMaxWaitTime property
 - in synchronous callbacks, 8-2

T

- task action time limits
 - specifying, 25-29, 25-33, 25-36
- task admin
 - definition, 24-7
- task assignments
 - dynamic, 24-6
 - rule-based, 24-6
 - static, 24-6
- task category
 - specifying, 25-10
- task conditions
 - abruptly completing a condition, 25-41
- task deadlines

- definition, 24-8
- task display form
 - creating, 26-6, 26-13, 26-15
 - definition, 25-3, 26-1
 - deploying, 26-25, 26-27
 - displaying, 26-35
- .task file
 - associating with a BPEL process, 25-2, 25-77
 - definition, 25-2, 25-5
- task flow
 - ADF
 - task display form for human tasks, 26-3
- task history
 - specifying in human task activities, 25-84
- task initiator
 - definition, 24-7
 - specifying, 25-80
- task instance attributes, 29-21
- task metadata service
 - definition, 24-28
 - Enterprise JavaBeans, SOAP, and Java support, 29-2
 - supported task operations, 29-11
 - WSDL file location, 29-2
- task notification
 - editing notification messages, 25-59
 - making email actionable, 29-30
 - notifying recipients of changes to task status, 25-57
 - overview, 25-56
 - reminders, 29-32
 - securing notifications, 29-32
 - setting up reminders, 25-60
 - task attachments with email notifications, 29-31
- task outcome
 - specifying, 25-8
- task owner
 - definition, 24-7
 - specifying by browsing the user directory, 25-11
 - specifying in human task activities, 25-83
 - specifying through XPath expressions, 25-15
- task parameters
 - specifying, 25-80
- task participants
 - allowing all participants to invite other participants, 25-41
 - assigning task participants by name or expression, 25-25, 25-52
 - bypassing, 25-30, 25-34, 25-37
 - dynamically assigning with the assignment service, 29-37
 - inviting additional task participants, 25-30, 25-34, 25-36
 - sharing attachments and comments, 25-33
 - specifying, 25-18
- task payload data structure
 - specifying, 25-16
- task priority
 - specifying, 25-10, 25-80
- task query service

- definition, 24-28
- Enterprise JavaBeans, SOAP, and Java support, 29-2
- supported task operations, 29-7
- WSDL file location, 29-2
- task reminders
 - setting up, 25-60
- task report service
 - Enterprise JavaBeans, SOAP, and Java support, 29-2
 - supported task operations, 29-14
 - WSDL file location, 29-3
- task reviewer
 - definition, 24-7
- task routing service
 - definition, 24-28
- task service
 - definition, 24-28
 - Enterprise JavaBeans, SOAP, and Java support, 29-2
 - supported task operations, 29-4
 - WSDL file location, 29-2
- task stages
 - definition, 24-9
- task title
 - specifying, 25-79
- tasks
 - escalating, renewing, or ending a task, 25-52
 - notifications and reminders, 29-25
- TCP tunneling
 - setting up a TCP listener for asynchronous services, 9-11
 - setting up a TCP listener for synchronous services, 9-11
- terminate activity
 - capabilities, A-21
 - definition, 12-33
 - fault handling, 12-33
- test suites
 - components, 49-2
 - creating, 49-5
 - definition, 49-1
 - limitations on multibyte character names, 49-5
- throw activity
 - capabilities, A-21
 - throwing internal faults, 12-22
- time
 - assigning with a function, 7-18
- time dimensions, 31-15
- time duration format, 14-2
- time stamp field, 31-5
- time zones, changing, 27-63
- timeouts
 - of BPEL processes, 14-1
 - using pick activities, 14-1
 - using the wait activity, 14-5
- timezone-from-dateTime function
 - description, B-7
- title
 - specifying, 25-8

- trackable fields
 - composite sensors, 51-1
- transform activity
 - capabilities, A-22
 - creating, 45-6
- transformations
 - adding XSLT constructs, 45-24
 - auto mapping, 45-30
 - auto mapping with confirmation, 45-32
 - chaining functions, 45-19
 - correcting memory errors, 45-47
 - creating, 45-6
 - creating a map file from imported schemas, 45-8
 - creating a new map file, 45-6
 - creating an XSL map from an XSL style sheet, 45-6
 - customizing sample XML generation, 45-48
 - dictionaries, 45-35
 - editing functions, 45-18
 - editing XPath expressions, 45-22
 - error when mapping duplicate elements, 45-6
 - functions, 45-17
 - functions prefixed with xp20 or orcl, 45-17
 - generating optional elements, 45-48
 - generating reports, 45-46
 - ignoring elements, 45-39
 - linking source target nodes, 45-16
 - map parameter and variable creation, 45-36
 - named templates in functions, 45-19
 - repeating elements, 45-27
 - replacing schemas, 45-39
 - rules, 45-6
 - searching source and target nodes, 45-38
 - setting constant values, 45-16
 - setting the maximum depth, 45-48
 - setting the number of repeating elements, 45-48
 - testing the map file, 45-43
 - using arrays, 45-27
 - using the XSLT Mapper, 45-14
 - using XQuery and XSLT, 7-4
 - viewing unmapped target nodes, 45-34
 - xsl choose conditional processing, 45-26
 - xsl if conditional processing, 45-25
- translateFromNative function
 - description, B-36
- translateToNative function
 - description, B-36

U

- UDDI *See* Oracle Service Registry
- undeployment
 - SOA composite applications, 4-26
- Unicode support, 4-4
- upper-case function
 - description, B-14
- user directory
 - selecting notification recipients by browsing the directory, 16-12
- user metadata service

- definition, 24-28
- Enterprise JavaBeans, SOAP, and Java support, 29-2
- supported task operations, 29-12
- WSDL file location, 29-2
- user notification activity
 - capabilities, A-23
- user notifications
 - definition, 16-13
- using domain value maps, 46-8
- using domain value maps a transformation, 46-9
- using error handling, 20-7
- using lookupValue functions, 46-11
- using Oracle Mediator error handling, 20-7

V

- validate syntax (XSD), 18-25
- validate syntax (XSD) property, 18-25
- validation
 - of XML data with bpelx
 - validate, 7-27
 - when loading a process diagram, A-32
- variable sensors
 - definition, 17-2
- variables
 - complex type, 7-15
 - copying data between, 7-14
 - initializing with expression constants, 7-13
 - initializing with literal XML, 7-13
- ViewController project, 30-2
- voice activity
 - capabilities, A-24
- voice mail
 - dynamically setting telephone numbers, 16-11
 - notifications support, 16-11

W

- wait activity
 - capabilities, A-24
 - code example, 14-5
 - definition, 14-5
- web service
 - DataObjectDefinition, 36-3
 - DataObjectOperations, 36-2
 - ICommand, 36-4
 - ManualRuleFire, 36-4
 - service component, 23-10
 - WSDL, 23-10
- WebLogic Fusion Order Demo application
 - introduction, 3-1
 - overview, 3-3
 - setting up, 3-2
- web.xml file, 30-3
- wfDynamicGroupAssign function
 - description, B-48
- wfDynamicUserAssign function
 - description, B-49
- while activity

- capabilities, A-25
- in conditional branching logic, 11-4
- wires
 - definition, 1-8
 - deleting, 4-16
 - using, 4-15
 - wiring a service component and reference, 4-19
- WLST utility
 - creating a configuration plan, 43-36
 - deployment with, 43-22
- WordML style sheets
 - using for attachments, 25-63
- workflow context
 - creating on behalf of a user, 29-4
- workflow functions
 - overview, 29-1
- workflow service clients, 28-3
 - interface, 28-5
- workflow services
 - abruptly completing a condition, 25-41
 - actionable emails, 29-30
 - allowing all participants to invite other participants, 25-41
 - assigning task participants by name or expression, 25-25, 25-52
 - assignment service configuration, 29-34
 - associating the human task activity with a BPEL process, 25-77
 - associating the human task definition with a BPEL process, 25-2
 - bypassing task participants, 25-30, 25-34, 25-37
 - editing notification messages, 25-59
 - Enterprise JavaBeans references, 28-7
 - Enterprise JavaBeans support, 29-1
 - escalate after policy, 25-55
 - escalating, renewing, or ending a task, 25-52
 - escalation and expiration policy overview, 25-52, 25-53
 - escalation rules, 25-62
 - expire after policy, 25-54
 - functions
 - clearTaskAssignees, B-45
 - createWordMLDocument, B-45
 - getNotificationProperty, B-45
 - getNumberOfTaskApprovals, B-46
 - getPreviousTaskApprover, B-46
 - getTaskAttachmentByIndex, B-46
 - getTaskAttachmentByName, B-47
 - getTaskAttachmentContents, B-47
 - getTaskAttachmentsCount, B-47
 - getTaskResourceBindingString, B-47
 - wfDynamicGroupAssign, B-48
 - wfDynamicUserAssign, B-49
 - FYI assignee task participant, 25-37
 - group voting details, 25-32
 - identification key, 25-83
 - identity service, 24-28
 - including the task history of other tasks, 25-84
 - inviting additional task participants, 25-30, 25-34, 25-36

- Java support, 29-1
- multilingual settings, 25-63, 29-29
- never expire policy, 25-53
- notification contents, 29-26
- notification preferences, 25-56
- notification service, 24-28, 29-28
- notifications, 29-25
- notifying recipients of changes to task
 - status, 25-57
- overview, 29-1
- parallel task participant, 25-30
- renew after policy, 25-54
- routing slip
 - definition, 25-29
- runtime config service, 24-29
- scope name and global task variable name, 25-83
- securing notifications, 29-32
- security model, 29-3, 29-4
- serial task participant, 25-34
- setting up reminders, 25-60
- sharing attachments and comments with task
 - participants, 25-33
- single approver task participant, 25-22
- SOAP support, 29-1
- specifying a task initiator and task priority, 25-80
- specifying a task title, 25-79
- specifying callback classes, 25-65
- specifying task parameters, 25-80
- style sheets in attachments, 25-63
- support for identity service, 29-9
- task attachments with email notifications, 29-31
- task category, 25-10
- task display form, 25-3, 26-1
- .task file
 - definition, 25-2, 25-5
- task metadata service, 24-28
- task notifications, 29-25
- task outcome, 25-8
- task owner, 25-83
- task owner specification through the user
 - directory, 25-11
- task owner specification through XPath
 - expressions, 25-15
- task participants, 25-18
- task payload data structure, 25-16
- task priority, 25-10
- task query service, 24-28
- task routing and customization in BPEL
 - callbacks, 25-76
- task routing service, 24-28
- task service, 24-28
- task title, 25-8
- time limits for acting on tasks, 25-29, 25-33, 25-36
- user metadata service, 24-28
- viewing BPEL callbacks, 25-87
- WordML style sheets in attachments, 25-63

worklist

- acting on tasks, 27-27
- acting on tasks that require a digital
 - signature, 27-34
- administration functions, 27-43
- approving tasks, 27-37
- assignment rules for tasks with multiple
 - assignees, 27-43
- changing the display, 27-44
- creating a subtask, 27-20
- creating a ToDo list, 27-19
- creating and customizing worklist views, 27-14
- creating group rules, 27-41
- creating user rules, 27-40
- customizing the task status chart, 27-18
- definition, 27-1
- filtering tasks, 27-7
- logging in, 27-3
- managing messaging channels, 27-47
- managing messaging filters, 27-49
- managing rules, 27-43
- mapping flex fields, 27-53
- messaging filter rules, 27-46
- reports, 27-56, 27-57
- rule actions, 27-47
- setting a vacation period, 27-38
- setting rules, 27-39
- specifying notification settings, 27-46
- system actions, 27-24
- Task Details page, acting on tasks, 27-21
- task history, 27-24
- Task Listing page contents, 27-6
- Task Listing page, customizing, 27-7
- using flex fields, 27-52

worklist clients

- building for workflow services, 28-1
- class paths for clients using local Enterprise
 - JavaBeans, 28-7
- class paths for clients using remote Enterprise
 - JavaBeans, 28-6
- class paths for clients using SOAP, 28-6
- customizing, 28-1
- packages and classes for, 28-2

writeBinaryToFile function

- description, B-31

WS-Addressing

- sending correlation IDs, 9-8

wsclient.jar file, 30-2

WSDL

- service component metadata, 23-10

WSDL files

- definition, 1-2, 2-6
- differences between document-literal styles and
 - RPC styles, 7-1, 7-33
- invoking the default revision, 4-15
- location for evidence store service, 29-3
- location for identity service, 29-2
- location for runtime config service, 29-3
- location for task metadata service, 29-2
- location for task query service, 29-2
- location for task report service, 29-3
- location for task service, 29-2
- location for user metadata service, 29-2
- modifying to generate a fault, 12-21

- references, 4-14
- selecting, 4-11
- specifying when creating a partner link, 5-6
- using an existing WSDL file, 4-12

X

XML assert

- overview, 49-2

XML data in BPEL, 7-2

XML data manipulation

- bpelx:append extension, 7-20
- bpelx:copyList extension, 7-26
- bpelx:insertAfter extension, 7-22
- bpelx:insertBefore extension, 7-21
- bpelx:remove extension, 7-23
- bpelx:rename extension, 7-24
- bpelx:validate extension, 7-27

XML documents

- manipulating, 7-2, 7-4
- overview, 7-2, 7-4

XML facades

- definition, 13-3
- Java embedding, 13-3

XML schema files

- error handling, 20-8
- fault-bindings.xml, 20-12
- fault-policies.xml, 20-8

XML schemas

- message types and variable types, 7-1

XPath expressions

- assigning numeric values, 7-16
- boolean expressions in switch activities, 11-3
- dynamically creating another XPath expression, 7-29
- dynamically setting email addresses and telephone numbers, 16-12
- editing in transformations, 45-22
- examples, 7-3
- fetching a data sequence element, 7-29
- in conditional branching logic, 11-1
- specifying a task owner, 25-15

XPath extension functions

- creating user-defined functions, B-59
- dvm
 - lookupValue function, 46-8
 - lookupValue1M function, 46-9

XPath functions

- examples, 7-4
- in transformations, 45-17
- indexing methods, 7-29
- mathematical calculations, 7-16

XPath queries

- copying data, 7-15
- examples, 7-3

XQuery, 7-2, 7-4

xref

- lookupXRef function, 47-12
 - exception reasons, 47-13
 - parameters, 47-12

lookupXRef1M function

- exception reasons, 47-14
- parameters, 47-13, 47-14

markForDelete function, 47-16

- exception reasons, 47-16
- parameters, 47-16

populateXRefRow function

- modes, 47-7
- parameters, 47-7

populateXRefRow1M function, 47-9

- modes, 47-9
- parameters, 47-9

xsl choose

- conditional processing, 45-26

xsl if

- conditional processing, 45-25

XSL map

- creating from an XSL style sheet, 45-6

XSL style sheet

- creating an XSL map, 45-6

XSL transformations

- definition, 1-2

XSLT, 7-2, 7-4

XSLT constructs

- adding in transformations, 45-24

XSLT Mapper

- adding XSLT constructs, 45-24
- auto mapping, 45-30
- auto mapping with confirmation, 45-32
- chaining functions, 45-19
- correcting memory errors when generating reports, 45-47
- creating a map file, 45-1
- creating a map file from imported schemas, 45-8
- creating a new map file, 45-6
- creating a transform activity, 45-6
- creating an XSL map from an XSL style sheet, 45-6
- customizing sample XML generation for transformations, 45-48
- dictionaries, 45-35
- editing functions, 45-18
- editing XPath expressions, 45-22
- error when mapping duplicate elements, 45-6
- functions, 45-17
- functions prefixed with xp20 or orcl, 45-17
- generating optional elements, 45-48
- generating reports, 45-46
- ignoring elements, 45-39
- layout in Oracle JDeveloper, 45-1
- linking source and target nodes, 45-16
- map parameter and variable creation, 45-36
- named templates in functions, 45-19
- repeating elements, 45-27
- replacing schemas, 45-39
- rules, 45-6
- searching source and target nodes, 45-38
- setting constant values, 45-16
- setting the maximum depth, 45-48
- setting the number of repeating elements, 45-48

- testing the map file, 45-43
- using, 45-14
- using arrays, 45-27
- viewing unmapped target nodes, 45-34
- xsl choose conditional processing, 45-26
- xsl if conditional processing, 45-25

XSLT mapper

- using, 19-49

Y

year-from-dateTime function

- description, B-7